AIMMS

The User's Guide

AIMMS 4

AIMMS

The User's Guide

AIMMS

Marcel Roelofs
Johannes Bisschop

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

ISBN 978–0–557–06360–4

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

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

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

About AIMMS

History

AIMMS was introduced as a new type of mathematical modeling tool in 1993—an integrated combination of a modeling language, a graphical user interface, and numerical solvers. AIMMS has proven to be one of the world's most advanced development environments for building optimization-based decision support applications and advanced planning systems. Today, it is used by leading companies in a wide range of industries in areas such as supply chain management, energy management, production planning, logistics, forestry planning, and risk-, revenue-, and asset- management. In addition, AIMMS is used by universities worldwide for courses in Operations Research and Optimization Modeling, as well as for research and graduation projects.

What is AIMMS*?*

AIMMS is far more than just another mathematical modeling language. True, the modeling language is state of the art for sure, but alongside this, AIMMS offers a number of advanced modeling concepts not found in other languages, as well as a full graphical user interface both for developers and end-users. AIMMS includes world-class solvers (and solver links) for linear, mixed-integer, and nonlinear programming such as BARON, CPLEX, CONOPT, GUROBI, KNITRO, PATH, SNOPT and XA, and can be readily extended to incorporate other advanced commercial solvers available on the market today. In addition, concepts as stochastic programming and robust optimization are available to include data uncertainty in your models.

Mastering AIMMS

Mastering AIMMS is straightforward since the language concepts will be intuitive to Operations Research (OR) professionals, and the point-and-click graphical interface is easy to use. AIMMS comes with comprehensive documentation, available electronically and in book form.

Types of AIMMS *applications*

AIMMS provides an ideal platform for creating advanced prototypes that are then easily transformed into operational end-user systems. Such systems can than be used either as

- stand-alone applications, or
- optimization components.

Stand-alone applications

Application developers and operations research experts use AIMMS to build complex and large scale optimization models and to create a graphical end-user interface around the model. AIMMS-based applications place the power of the most advanced mathematical modeling techniques directly into the hands of end-users, enabling them to rapidly improve the quality, service, profitability, and responsiveness of their operations.

Optimization components

Independent Software Vendors and OEMs use AIMMS to create complex and large scale optimization components that complement their applications and web services developed in languages such as C++, Java, .NET, or Excel. Applications built with AIMMS-based optimization components have a shorter time-to-market, are more robust and are richer in features than would be possible through direct programming alone.

AIMMS *users*

Companies using AIMMS include

- ABN AMRO
- Areva
- Bayer
- Bluescope Steel
- BP
- CST
- ExxonMobil
- Gaz de France
- Heineken
- Innovene
- Lufthansa
- Merck
- Owens Corning
- Perdigão
- Petrobras
- Philips
- PriceWaterhouseCoopers
- Reliance
- Repsol
- Shell
- Statoil
- Unilever

Universities using AIMMS include Budapest University of Technology, Carnegie Mellon University, George Mason University, Georgia Institute of Technology, Japan Advanced Institute of Science and Technology, London School of Economics, Nanyang Technological University, Rutgers University, Technical University of Eindhoven, Technische Universität Berlin, UIC Bioengineering, Universidade Federal do Rio de Janeiro, University of Groningen, University of Pittsburgh, University of Warsaw, and University of the West of England.

A more detailed list of AIMMS users and reference cases can be found on our website www.aimms.com.

Contents

Preface

Three AIMMS *books*

The printed AIMMS documentation consists of three books

- AIMMS—*The User's Guide*,
- AIMMS—*The Language Reference*, and
- AIMMS—*Optimization Modeling*.

The first two books emphasize different aspects in the use of the AIMMS system, while the third book is a general introduction to optimization modeling. All books can be used independently.

Available online

In addition to the printed versions, these books are also available on-line in the ADOBE Portable Document Format (PDF). Although new printed versions of the documentation will become available with every new functional AIMMS release, small additions to the system and small changes in its functionality in between functional releases are always directly reflected in the online documentation, but not necessarily in the printed material. Therefore, the online versions of the AIMMS books that come with a particular version of the system should be considered as the authoritative documentation describing the functionality regarding that particular AIMMS version.

Release notes

Which changes and bug fixes are included in particular AIMMS releases are described in the associated release notes.

What's new in AIMMS 4

From AIMMS 4.1 onwards, we will only publish this ”What's New” section on our website. It can be found at the following location:

https://aimms.com/english/developers/downloads/product-information/new-features/

What is in the Aimms documentation

The User's Guide

The Aimms User's Guide provides a global overview of how to use the Aimms system itself. It is aimed at application builders, and explores Aimms' capabilities to help you create a model-based application in an easy and maintainable manner. The guide describes the various graphical tools that the Aimms system offers for this task. It is divided into five parts.

- Part I—*Introduction to* Aimms—what is Aimms and how to use it.
- Part II—*Creating and Managing a Model*—how to create a new model in Aimms or manage an existing model.
- Part III—*Creating an End-User Interface*—how to create an intuitive and interactive end-user interface around a working model formulation.
- Part IV—*Data Management*—how to work with cases and datasets.
- Part V—*Miscellaneous*—various other aspects of Aimms which may be relevant when creating a model-based end-user application.

The Language Reference

The Aimms Language Reference provides a complete description of the Aimms modeling language, its underlying data structures and advanced language constructs. It is aimed at model builders only, and provides the ultimate reference to the model constructs that you can use to get the most out of your model formulations. The guide is divided into seven parts.

- Part I—*Preliminaries*—provides an introduction to, and overview of, the basic language concepts.
- Part II—*Nonprocedural Language Components*—describes Aimms' basic data types, expressions, and evaluation structures.
- Part III—*Procedural Language Components*—describes Aimms' capabilities to implement customized algorithms using various execution and flow control statements, as well as internal and external procedures and functions.
- Part IV—*Sparse Execution*—describes the fine details of the sparse execution engine underlying the Aimms system.
- Part V—*Optimization Modeling Components*—describes the concepts of variables, constraints and mathematical programs required to specify an optimization model.
- Part VI—*Data Communication Components*—how to import and export data from various data sources, and create customized reports.
- Part VII—*Advanced Language Components*—describes various advanced language features, such as the use of units, modeling of time and communicating with the end-user.

Optimization Modeling

The book on optimization modeling provides not only an introduction to modeling but also a suite of worked examples. It is aimed at users who are new to modeling and those who have limited modeling experience. Both basic concepts and more advanced modeling techniques are discussed. The book is divided into five parts:

- Part I—*Introduction to Optimization Modeling*—covers what models are, where they come from, and how they are used.
- Part II—*General Optimization Modeling Tricks*—includes mathematical concepts and general modeling techniques.
- Part III—*Basic Optimization Modeling Applications*—builds on an understanding of general modeling principles and provides introductory application-specific examples of models and the modeling process.
- Part IV—*Intermediate Optimization Modeling Applications*—is similar to part III, but with examples that require more effort and analysis to construct the corresponding models.
- Part V—*Advanced Optimization Modeling Applications*—provides applications where mathematical concepts are required for the formulation and solution of the underlying models.

Documentation of deployment features

In addition to the three major Aimms books, there are several separate documents describing various deployment features of the Aimms software. They are:

- Aimms—*The Function Reference*,
- Aimms—*The COM Object User's Guide and Reference*,
- Aimms—*The Excel Add-In User's Guide*, and
- Aimms—*The Open Solver Interface User's Guide and Reference*.

These documents are only available in PDF format.

Help files

The Aimms documentation is complemented with a number of help files that discuss the finer details of particular aspects of the Aimms system. Help files are available to describe:

- the execution and solver options which you can set to globally influence the behavior of the Aimms' execution engine,
- the finer details of working with the graphical modeling tools, and
- a complete description of the properties of end-user screens and the graphical data objects which you can use to influence the behavior and appearance of an end-user interface built around your model.

The Aimms help files are both available as Windows help files, as well as in PDF format.

Aimms *tutorials*

Two tutorials on Aimms in PDF format provide you with some initial working knowledge of the system and its language. One tutorial is intended for beginning users, while the other is aimed at professional users of Aimms.

Searching the documentation

As the entire Aimms documentation is available in PDF format, you can use the search functionality of Acrobat Reader to search through all Aimms documentation for the information you are looking for.

Aimms *model library*

Aimms comes with an extensive model library, which contains a variety of examples to illustrate simple and advanced applications containing particular aspects of both the language and the graphical user interface. You can find the Aimms model library in the Examples directory in the Aimms installation directory. The Examples directory also contains an Aimms project providing an index to all examples, which you can use to search for examples that illustrate specific aspects of Aimms.

What is in the User's Guide

Introduction to Aimms

Part I of the User's Guide provides a basic introduction to Aimms, its position among other technologies, and its use.

- Chapter 1—Aimms *and Analytic Decision Support*—discusses the concept of Analytic Decision Support (ADS), Aimms as an ADS development environment, as well as a comparison to other ADS tools.
- Chapter 2—*Getting Started*—explains how to create a new Aimms application, and provides an overview of both the modeling tools available in Aimms and the files associated with an Aimms project.
- Chapter 3—*Organizing a Project into Libraries*—describes the facilities available in Aimms to allow multiple developers to collaborate on a single project.

Creating and managing a model

Part II discusses all aspects of the Aimms system that are relevant for entering and maintaining the model source associated with a particular modeling application.

- Chapter 4—*The Model Explorer*—introduces the main graphical tool available in Aimms for accessing the model source. It discusses various aspects that are specific to the model tree, as well as the basic concepts common to all trees used in the Aimms system.
- Chapter 5—*Identifier Declarations*—explains how you can add identifier declarations to the model tree, and how you can modify the various attributes of an identifier in its attribute window.
- Chapter 6—*Procedures and Functions*—explains how you can create procedures and functions within your model, how to add arguments to such procedures and functions, and describes the Aimms concepts that

help you to sub-divide procedure and function bodies into smaller more meaningful entities.
- Chapter 7—*Viewing Identifier Selections*—discusses the flexible identifier selector tool in AIMMS, which allows you to create and simultaneously view selections of identifiers in your model.
- Chapter 8—*Debugging and Profiling an* AIMMS *Model*—discusses AIMMS' debugger and profiler, which can help you to track the modeling errors in an AIMMS model, or to find and speed up time-consuming statements in your model.
- Chapter 9—*The Math Program Inspector*—introduces a graphical debugging tool for finding infeasibilities or unexpected results of a math program contained in your model.

Creating an end-user interface

Part III introduces the fundamental concepts and design tools available in AIMMS to create a graphical end-user interface for your modeling application, as well as AIMMS' reporting facilities.

- Chapter 10—*Pages and Page Objects*—introduces the AIMMS concept of end-user pages. In addition, it explains how to add graphical (data) objects to such pages, and how to link these data objects to identifiers in your model.
- Chapter 11—*Page and Page Object Properties*—discusses the options for pages and page objects that you can modify to alter the behavior and appearance of your end-user interface.
- Chapter 12—*Page Management Tools*—describes the AIMMS tools that can help you create and manage a large collection of end-user pages in an easily maintainable fashion.
- Chapter 13—*Page Resizability*—explains the basic concepts available in AIMMS to define the behavior of pages when resizing.
- Chapter 14—*Creating Printed Reports*—discusses the concept of print pages which you can use to create a printed report of your model results.
- Chapter 15—*Deploying End-User Applications*—discusses the several options that AIMMS has to offer to deploy your AIMMS applications.

Data management

Part IV focuses on the facilities within AIMMS for performing case management tasks. Chapter 16—*Case Management*—describes the case management facilities and tools in AIMMS. It also provides you with an overview of AIMMS' capabilities to work with multiple case data, both in the model and the end-user interface.

Miscellaneous

Part V discusses the various miscellaneous concepts that may be of interest to both AIMMS developers and/or end-users.

- Chapter 17—*User Interface Language Components*—provides a complete overview of the function library available in AIMMS for communication

with the end-user through the various tools available in the Aimms end-user interface.
- Chapter 18—*Calling* Aimms—describes Aimms' command line options, the restrictions with respect to end-user licenses, and the possibilities of calling an Aimms model from within your own application.
- Chapter 19—*Project Security*—discusses various security aspects such as encrypting your project, and adding a user database to a model to provide user authentication.
- Chapter 20—*Project Settings and Options*—describes the tools available in Aimms to alter the execution behavior of your model, the appearance of its interface, and various other aspects concerning Aimms itself and its solvers.
- Chapter 21—*Localization Support*—discusses Aimms' built-in support for localizing the end-user interface of your project (i.e. making it capable of dealing with multiple languages).

The authors

Marcel Roelofs

Marcel Roelofs received his Ph.D. in Applied Mathematics from the Technical University of Twente in 1993 on the application of Computer Algebra in Mathematical Physics. From 1993 to 1995 he worked as a post-doc at the Centre for Mathematics and Computer Science (CWI) in Amsterdam in the area of Computer Algebra, and had a part-time position at the Research Institute for the Application of Computer Algebra. In 1995 he accepted his current position as CTO of AIMMS B.V. His main responsibilities are the design and documentation of the Aimms language and user interface.

Johannes Bisschop

Johannes Bisschop received his Ph.D. in Mathematical Sciences from the Johns Hopkins University in Baltimore USA in 1974. From 1975 to 1980 he worked as a Researcher in the Development Research Center of the World Bank in Washington DC, USA. In 1980 he returned to The Netherlands and accepted a position as a Research Mathematician at Shell Research in Amsterdam. After some years he also accepted a second part-time position as a full professor in the Applied Mathematics Department at the Technical University of Twente. From 1989 to 2003 he combined his part-time position at the University with managing Paragon Decision Technology B.V. and the continuing development of Aimms. From 2003 to 2005 he held the position of president of Paragon Decision Technology B.V. His main interests are in the areas of computational optimization and modeling.

Other contributors to Aimms

In addition to the main authors, various current and former employees of Aimms B.V. (formerly known as Paragon Decision Technology B.V.) and external consultants have made a contribution to the Aimms documentation. They are (in alphabetical order):

- Pim Beers
- John Boers
- Peter Bonsma
- Mischa Bronstring
- Ximena Cerda Salzmann
- Michelle Chamalaun
- Horia Constantin
- Guido Diepen
- Robert Entriken
- Floor Goddijn
- Thorsten Gragert
- Koos Heerink
- Nico van den Hijligenberg
- Marcel Hunting
- Roel Janssen
- Gertjan Kloosterman
- Joris Koster
- Chris Kuip
- Gertjan de Lange
- Ovidiu Listes
- Peter Nieuwesteeg
- Franco Peschiera
- Bianca Rosegaar
- Diego Serrano
- Giles Stacey
- Richard Stegeman
- Selvy Suwanto
- Jacques de Swart
- Martine Uyterlinde

Part I

Introduction to AIMMS

Chapter 1

Aimms and Analytic Decision Support

What is Aimms*?*

The acronym Aimms stands for

> *Advanced Integrated Multidimensional Modeling Software.*

Aimms offers you an easy-to-use and all-round development environment for creating fully functional *Analytic Decision Support* (ADS) applications ready for use by end-users. The software is constructed to run in different modes to support two primary user groups: *modelers* (application developers) and *end-users* (decision makers). Aimms provides the ability to place all of the power of the most advanced mathematical modeling techniques directly into the hands of the people who need this to make decisions.

This chapter

This chapter is aimed at first-time users of the Aimms modeling system. In a nutshell, it provides

- a description of the characteristics of Analytic Decision Support (ADS) applications,
- an overview of Aimms as an ADS development environment, and
- some examples of its use in real-life applications.

1.1 Analytic decision support

Analytic decision support

Analytic decision support applications are usually interactive decision support systems with a strong internal emphasis on *advanced computational techniques* and that pertain to extensive problem analysis on the outside. They typically

- represent a complex and large-scale reality,
- organize and use large amounts of interrelated multidimensional data based on corporate and market information,
- use advanced arithmetic manipulations and/or optimization tools to find a solution,
- apply analytic techniques or perform “what-if” experiments to assess the consequences of making a decision under different scenarios,
- employ advanced visualization techniques to provide an insight into the solution and/or the problem complexity, and

- are subject to permanent change due to a changing reality or improved insights.

Increasing market need

With the world becoming daily more complex, decision makers around the world are in search of advanced decision support tools. Such tools can help them get insights into their decision problems, monitor the consequences of previous decisions, and help them take new decisions on a regular basis. There is substantial evidence that analytic decision support applications are becoming increasingly popular throughout industry and government, as the improved decisions generated by ADS applications imply increased profit and/or efficiency.

Supporting developments

A number of major developments in the last decade have increased the suitability of analytic decision support to tackle such problems:

- corporate databases are becoming increasingly mature and allow a quick follow-up to market changes,
- the increasing speed of PCs allows interactive use, even with complex applications,
- the visually attractive and convenient presentation using the standardized and user-friendly Windows environment makes complex processes more accessible to decision makers, and
- the availability of standardized and improved optimization tools allows ADS application developers to specify the problem without having to specify a complicated algorithm to solve it.

Applicable problem areas

Analytic decision support lends itself to a wide variety of decision support problems. The following list provides a non-exhaustive overview of the areas in which analytic decision support is applicable:

- strategic and tactical planning of resources in industry and government,
- operational scheduling of machines, vehicles, product flow and personnel,
- strategic evaluation studies in the areas of energy, environment, forestry and social policies,
- financial decision-making to support asset-liability management,
- economic decision-making to control market clearing and economic development, and
- technical decision-making to support the design and calibration of systems and objects.

1.2 AIMMS as an ADS development environment

AIMMS *as ADS development environment*

As an ADS development environment, AIMMS possesses a unique combination of advanced features and design tools which allow you to build complex ADS applications which are easily maintainable—in a fraction of the time required with conventional programming languages. Figure 1.1 provides a top-level overview of the components available in AIMMS.

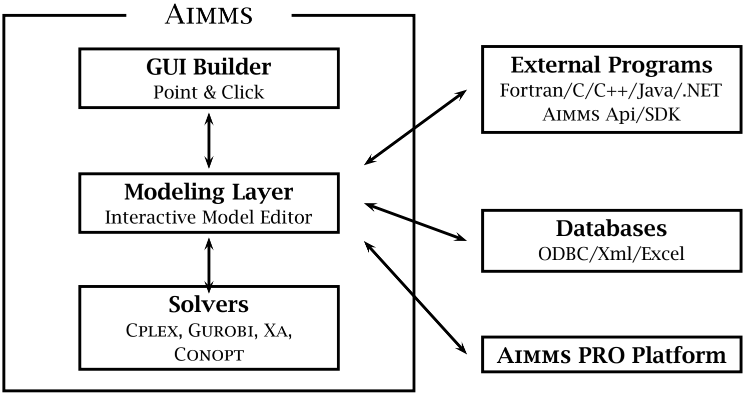


Figure 1.1: Graphical overview of AIMMS components

Multidimensional modeling language

The multidimensional modeling language in AIMMS offers a powerful index notation which enables you to capture the complexity of real-world problems in an intuitive manner. In addition, the language allows you to express very complex relationships in a compact manner without the need to worry about memory management or sparse data storage considerations. The combined declarations and procedures using these multidimensional structures can be organized, edited and displayed using an advanced interactive model editor.

Optimization modeling

One of the outstanding features of AIMMS is the capability of specifying and solving linear and nonlinear constraint-based optimization models. Using the same compact and rich notation available for procedural statements, symbolic constraints can be formulated in a simple and concise manner. With only a single instruction, an optimization model can be transferred to, and solved by, world- class solvers such as CPLEX, GUROBI and CONOPT.

Advanced language features

Selected advanced AIMMS language features include:

- a rich set of mathematical, statistical and financial functions,
- a powerful combination of (automatically updated) multidimensional definitions and procedural execution,
- the ability to easily express time-based models through the use of calendars and horizons, including support for rolling horizons with automatic aggregation and disaggregation, and
- the ability to associate units of measurement with model identifiers assuring unit consistency within expressions.

Integrated GUI builder

In addition to its versatile modeling language AIMMS offers an integrated tool for constructing a custom graphical user interface (GUI) for your decision support application. End-user screens can be created in an easy point-and-click manner, and can include such common graphical objects as tables, charts and curves, all closely linked to multidimensional identifiers in your model. Included, amongst other more advanced objects, are a Gantt chart for visualizing time-phased planning/scheduling applications, and a network flow object for visualizing two-dimensional maps and flows.

Advanced GUI tools

To support you in creating complete end-user interfaces in a quick and maintainable fashion, AIMMS offers the following advanced tools:

- the *template manager* enables you to create a uniform look and feel by allowing common objects (such as headers, footers, and navigation buttons) to be placed on hierarchically organized templates which can be inherited by multiple pages,
- the *page manager* allows you to specify a natural page order, with which you can guide an end-user through your application by adding special page manager-driven navigation controls to templates or pages,
- the *menu builder* enables you to create customized end-user menus and toolbars to be added to your templates and pages,
- the *identifier selection wizard* assists you not only in selecting complete model identifiers, or slices thereof, for graphical display, but also in quickly linking data from various page objects.

Integrated case management

Case management forms an important part of any decision support application, and enables end-users to run the model with varying scenarios. AIMMS also offers advanced data management, which allows you to create data categories for holding blocks of related data (for instance topology data, or supply and demand scenarios). Data sets associated with these data categories can be combined to form a single case, and thus can be shared by more than one case. In addition, to perform an extensive what-if analysis, you can select a large number of cases and run them in batch mode overnight.

Database connectivity

As data form the life blood of any decision support application, AIMMS offers extensive facilities to link your application to corporate databases using ODBC. Specialized wizards help you relate columns in a database table with the corresponding multidimensional identifiers in your AIMMS model. Once you have established such relationships, you can specify straightforward read and write statements to transfer data to and from the database.

Linkages to other applications

To facilitate the re-use of existing code, or to speed up computationally intensive parts of your application, AIMMS allows you to execute external procedures or functions in a DLL from within your model. External functions can even be used within the constraints of an optimization model. In addition, AIMMS offers an in-process Application Programming Interface (API) for C/C++, as well as the out-of-process AIMMS SDK for Java/.Net/C++ which enables you to use your AIMMS model as a component from within an external application, to communicate data in a sparse fashion, and to execute procedures written in AIMMS.

User management

The AIMMS system has integrated facilities to create a database of end-users and link this database to one or more AIMMS-based applications. The end-user database contains information on the level of authorization of all end-users within an application. Through these authorization levels you can specify whether an end-user is allowed to access case data, view pages, modify data, and execute particular parts of the model.

Protecting your investment

The development of a professional decision support application usually represents a considerable investment in time and thus money. AIMMS offers facilities to protect this investment and to prevent unauthorized use of particular applications. Your project and the source code of your model can be encrypted using either a password or key based encryption scheme.

Extensive documentation

AIMMS comes complete with extensive documentation in the form of three books:

- a User's Guide to explain the overall functionality and use of AIMMS,
- a Language Reference giving a detailed description of the AIMMS language, and
- a Modeling Guide introducing you to both basic and advanced modeling techniques.

All of these books are available in hard copy as well as in electronic form. In addition, each system comes complete with a collection of example applications elucidating particular aspects of the language and end-user interface.

1.3 What is AIMMS used for?

AIMMS usage

AIMMS is used worldwide as a development environment for all kinds of analytic decision support applications. To give you some insight into the areas in which AIMMS has been used successfully, this section describes a small subset of ADS applications, namely those in which AIMMS itself has been involved (sometimes actively, sometimes at a distance).

Crude oil scheduling

The crude oil scheduling system covers the allocation, timetabling, blending and sequencing activities from the waterfront (arrival of crude ships) via the crude pipeline to the crude distillation units. The result is a schedule for the discharge of crudes, the pipeline and the crude distillers (sequencing, timing and sizing of crude batches), plus planning indications on the arrival of new crude deliveries. Enhanced decision support includes improved and timely reaction to changes and opportunities (e.g. distressed crude cargoes, ship and pumping delays, operation disturbances) and improved integration between crude acquisition and unit scheduling.

Strategic forest management

The strategic forest management system allows foresters to interactively represent large forested areas at a strategic level. Such a flexible decision framework can help in understanding how a forest develops over time. The system also allows one to explore forest management objectives and their trade-offs, plus the preparation of long-term forest management plans.

Transport scheduling in breweries

The transport scheduling system for breweries allows end-users to interactively steer the flow of products through all phases of the brewing process from hops to bottled beer. The application can be used either in an automatic mode where the flow of products is totally determined by the system, or it can be used in a manual mode where the user can set or alter the flow using the Gantt chart. The system can also be used in a simulation mode to test the response of the entire brewery to varying demands over a longer period of time.

Risk management

The risk management system for market makers and option traders has a wide functionality including the theoretical evaluation of derivatives, an extensive sensitivity analysis, the display of risk profiles, the generation of scenarios, the generation of price quotes and exercise signals, minimization of risk exposure, the calculation of exercise premiums and implied data (volatilities and interest rates), plus an overview of all transactions for any day.

Refinery blending

The refinery blending system is a blend scheduling and mixture optimization system. It is able to handle the complete pooling and blending problem, and optimizes both the blend schedules and the mixes under a complete set of (real- life) operational constraints. The system offers user flexibility in that the user can decide upon the number of components, fuel mixtures, long versus short term scheduling, and stand-alone versus refinery-wide scheduling.

Catalytic cracker support

Catalytic cracking refers to a refining process in which hydrocarbons are converted into products with a lower molecular mass. The catalytic cracking support system has three major components: (a) a graphical user interface consisting of interactive pages, validation routines, plus reporting and data handling facilities, (b) the model equations, including those for heat, yields, product properties, economics, and (c) an on-line process control environment with an off-line mode in which multiple studies with differing parameters and variables can be compared.

Data reconciliation

Data reconciliation is the process of making the smallest possible adjustment to a collection of measurements within a system such that the adjusted data values satisfy all the balance constraints applicable to the system. In the particular application in question, data reconciliation was applied to a chemical process, requiring that the relevant mass, component and thermodynamic balances be satisfied for all units within the system.

1.4 Comparison with other ADS tools

ADS development tools

There are several tools available in the market that can, in principle, be used as a development environment for analytic decision support applications. The most well-known are:

- spreadsheets,
- databases,
- programming languages, and
- multidimensional modeling languages.

Comparison

Spreadsheets, databases and programming languages all have their strengths as development tools for a large variety of applications. Advanced modeling systems such as Aimms should not be seen as a complete replacement for these three development environments, but rather as a tool specifically designed for developing analytic decision support applications. The following paragraphs outline the advantages and disadvantages of each of these tools with respect to their suitability as a development environment for ADS.

Spreadsheet

If you are a fervent spreadsheet user, it seems only natural to build your ADS applications on top of a spreadsheet. However, this may not always be the best choice. A spreadsheet approach works well when:

- you don't need to specify a large number of relationships,
- there are only a few procedures to be written,
- the size of your data sets remains stable,
- the need to add or remove dimensions is limited, and
- you will carry out all the maintenance activities yourself.

When this is not the case, the AIMMS approach may offer a suitable alternative, because:

- specifying a large number of (often similar) relationships can be done using indexed identifiers and definitions for these identifiers,
- adding and managing both internal and external procedures is a straightforward task using the AIMMS language and model editor,
- modifying the size of any (index) set in AIMMS is natural, as there is a complete separation between structure and data,
- adding or removing dimensions takes place in the language and does not require the copying of cells or creating more worksheets, and
- not only can the structure of the entire model be made visible, but also the model editor allows someone else to create customized overviews of model structure for further maintenance.

Database

If you are a fervent database user, it seems only natural to build your ADS applications using a language such as Visual-C/C++, Delphi or PowerBuilder on top of a database such as Microsoft Access, and Oracle. However, this may not always be the best choice. Using a database approach works well when:

- all of the data for your application is already stored in a database,
- the end-user GUI requires relatively little programming,
- speed of data transfer is not crucial,
- there is a limited need to link to external solvers, and
- maintenance is carried out by yourself or another experienced programmer.

When this is not the case, the AIMMS approach may offer a suitable alternative, because:

- data transfer works well not only for data stored in a database, but also for data in text and case files,
- the compact modeling language combined with the point-and-click GUI builder minimizes the amount of programming required,
- internal data transfer during (the sparse) execution is extremely fast and does not require the repeated transfer of data between external programs,
- standard links to solvers are built into AIMMS, and
- compact and simple data structures on the one hand, and point-and-click GUI construction on the other hand, help ease maintenance.

Programming language

If you are a fervent programmer, it seems only natural to build your ADS applications using languages such as C/C++ or Fortran. However, this may not always be the best choice. Using a programming language works well when:

- efficient data structures require relatively little effort,
- there are many procedures to be written,
- development times are not crucial,
- there is a limited need to link to external programs, and
- maintenance is carried out by yourself or another experienced programmer.

When this is not the case, the AIMMS approach may offer a suitable alternative, because:

- the standard multidimensional data structures in AIMMS require no special effort, and are efficient since all data storage and data manipulations are carried out in a sparse manner,
- writing procedures in AIMMS is at least as easy as in a programming language: their control structures are similar, and AIMMS has the advantage that no special data structures are required,
- specially developed tools for the construction of programs and GUIs minimize development time,
- standard links to databases and solvers are built into AIMMS, and
- compact and simple data structures on the one hand, and point-and-click GUI construction on the other, help to ease maintenance.

Comparison summary

Table 1.1 summarizes the most important issues that determine the suitability of the above development tools as a development environment for ADS applications. The table focuses on

- the initial development effort to create an ADS application,
- the subsequent time required for product maintenance (extremely important due to the permanently changing nature of ADS applications), and
- the openness of the environment with respect to data entry formats and third party components.

A '+' indicates that the product scores well in this area, a '–' indicates that it does not perform well in this area.

Building tool	**Development time**	**Maintenance time**	**Openness**	**Suitability as an ADS tool**
Spreadsheet	+	––	++	+
Database	+	–	+	+
Programming language	–	–	++	++
Aimms	++	++	+	++

Table 1.1: Comparison of ADS development tools

Developer quote

In support of the comparison in Table 1.1, the following quote, from one of our customers, clearly expresses the advantages of using Aimms as a development environment for ADS applications.

> *"Software development requires four tasks: definition, design, implementation and testing. When using* Aimms*, the focus is on definition. The result is an implementation which can be immediately tested. I now spend the majority of my time working on the customer's problem, and verifying that we have got the requirements correct. My job is now that of an applications engineer, rather than a software engineer. One of our customers stated that our recent project with them (using* Aimms*) was the first software project in their history not to have a single 'Software Functionality Problem Report' generated."*

Chapter 2

Getting Started

This chapter

This chapter provides pointers to the various AIMMS examples that are available online and may help you to get a quick feel for the system. It explains the principle steps involved in creating a new AIMMS project, and it provides you with an overview of the various graphical tools available in AIMMS to help you create a complete Analytic Decision Support application. In addition, the chapter discusses the files related to an AIMMS project.

2.1 Getting started with AIMMS

Learn by example

For most people, learning to use a new software tool like AIMMS is made substantially easier by first getting to see a few examples of its use. In this way you can get a quick feel for the AIMMS system, and begin to form a mental picture of its functionality.

Getting started quickly

In addition, by taking one or more illustrative examples as a starting point, you are able to quickly create simple but meaningful AIMMS projects on your own, without having to read a large amount of documentation. Building such small projects will further enhance your understanding of the AIMMS system, both in its use and in its capabilities.

AIMMS *tutorial*

To get you on your way as quickly as possible, the AIMMS system comes with a tutorial consisting of

- a number of live demos illustrating the basic use of AIMMS,
- an extensive library of small example projects each of which demonstrates a particular component of either the AIMMS language or the end-user interface,
- a number of complete AIMMS applications, and
- worked examples corresponding to chapters in the book on optimization modeling.

Example projects

The library of small example projects deals with common tasks such as

- creating a new project,
- building a model with your project,
- data entry,
- visualizing the results of your model,
- case management, and
- various tools, tips and tricks that help you to increase your productivity.

What you learn

By quickly browsing through these examples, you will get a good understanding of the paradigms underlying the AIMMS technology, and you will learn the basic steps that are necessary to create a simple, but fully functional modeling application.

This User's Guide

Rather than providing an introduction to the use of AIMMS, the User's Guide deals, in a linear and fairly detailed fashion, with all relevant aspects of the use of AIMMS and its modeling tools that are necessary to create a complete modeling application. This information enables you to use AIMMS to its full potential, and provides answers to questions that go beyond the scope of the example projects.

The Language Reference

The Language Reference deals with every aspect of AIMMS data types and the modeling language. You may need this information to complete the attribute forms while adding new identifiers to your model, or when you specify the bodies of procedures and functions in your model.

The Optimization Modeling guide

The Optimization Modeling guide provides you with the basic principles of optimization modeling, and also discusses several advanced optimization techniques which you may find useful when trying to accomplish nontrivial optimization tasks.

How to proceed

The following strategy may help you to use AIMMS as efficiently and quickly as possible.

- Study some of the working examples to get a good feel for the AIMMS system.
- Select an example project that is close to what you wish to achieve, and take it as a starting point for your first modeling project.
- Consult any of the three AIMMS books whenever you need more thorough information about either the use of AIMMS, its language or tips on optimization modeling.

2.2 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

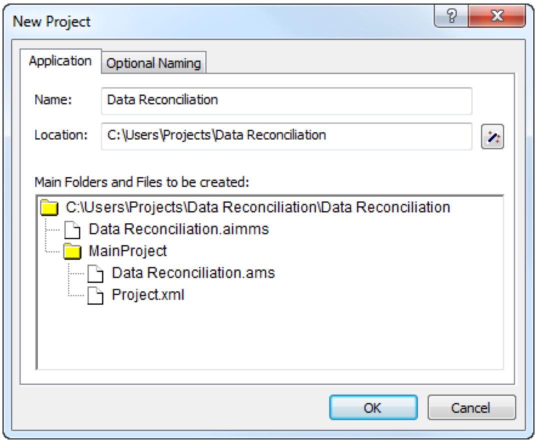
Within an AIMMS session, you can create a new project through the **File-New Project** menu. Note that this menu is only available when no other project is currently open. It will open the AIMMS **New Project** wizard illustrated in Figure 2.1. In this wizard you can enter the name of the new project, 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.

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.



Name: Data Reconciliation

Location: C:\Users\Projects\Data Reconciliation

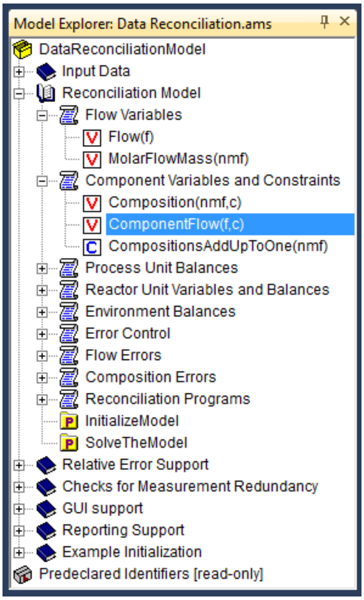
Main Folders and Files to be created:

- C:\Users\Projects\Data Reconciliation\Data Reconciliation
  - Data Reconciliation.aimms
  - MainProject
    - Data Reconciliation.ams
    - Project.xml

Figure 2.1: The AIMMS **New Project** wizard

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 Figure 2.2. 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



- DataReconciliationModel
  - Input Data
  - Reconciliation Model
    - Flow Variables
      - Flow(f)
      - MolarFlowMass(nmf)
    - Component Variables and Constraints
      - Composition(nmf,c)
      - ComponentFlow(f,c)
      - CompositionsAddUpToOne(nmf)
    - Process Unit Balances
    - Reactor Unit Variables and Balances
    - Environment Balances
    - Error Control
    - Flow Errors
    - Composition Errors
    - Reconciliation Programs
    - InitializeModel
    - SolveTheModel
  - Relative Error Support
  - Checks for Measurement Redundancy
  - GUI support
  - Reporting Support
  - Example Initialization
- Predeclared Identifiers [read-only]

Figure 2.2: The AIMMS Model Explorer

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 Section 2.3.

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.

2.3 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.

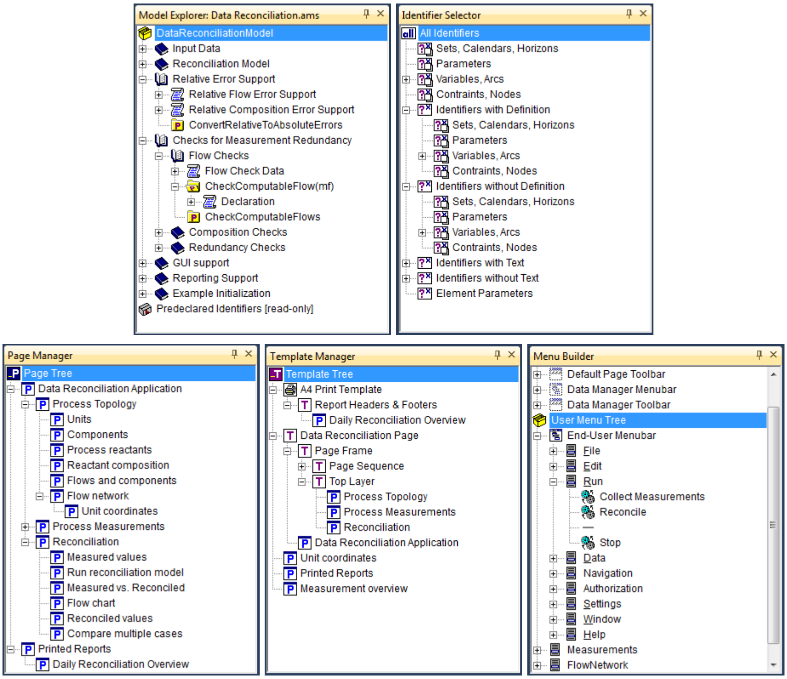
These tools can be accessed either through the **Tools** menu or via the project toolbar. They are all aimed at reducing the amount of work involved in developing, modifying and maintaining particular aspects of your model-based end-user application. Figure 2.3 provides an overview of the windows associated 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 Chapter 4.

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



Model Explorer: Data Reconciliation.ams

- DataReconciliationModel
  - Input Data
  - Reconciliation Model
  - Relative Error Support
    - Relative Flow Error Support
    - Relative Composition Error Support
    - ConvertRelativeToAbsoluteErrors
  - Checks for Measurement Redundancy
    - Flow Checks
      - Flow Check Data
      - CheckComputableFlow(mf)
        - Declaration
      - CheckComputableFlows
    - Composition Checks
    - Redundancy Checks
  - GUI support
  - Reporting Support
  - Example Initialization
- Predeclared Identifiers [read-only]

Identifier Selector

- All Identifiers
  - Sets, Calendars, Horizons
  - Parameters
  - Variables, Arcs
  - Contraints, Nodes
  - Identifiers with Definition
    - Sets, Calendars, Horizons
    - Parameters
    - Variables, Arcs
    - Contraints, Nodes
  - Identifiers without Definition
    - Sets, Calendars, Horizons
    - Parameters
    - Variables, Arcs
    - Contraints, Nodes
  - Identifiers with Text
  - Identifiers without Text
  - Element Parameters

Page Manager

- Page Tree
  - Data Reconciliation Application
    - Process Topology
      - Units
      - Components
      - Process reactants
      - Reactant composition
      - Flows and components
      - Flow network
        - Unit coordinates
    - Process Measurements
    - Reconciliation
      - Measured values
      - Run reconciliation model
      - Measured vs. Reconciled
      - Flow chart
      - Reconciled values
      - Compare multiple cases
  - Printed Reports
    - Daily Reconciliation Overview

Template Manager

- Template Tree
  - A4 Print Template
    - Report Headers & Footers
      - Daily Reconciliation Overview
  - Data Reconciliation Page
    - Page Frame
      - Page Sequence
      - Top Layer
        - Process Topology
        - Process Measurements
        - Reconciliation
    - Data Reconciliation Application
  - Unit coordinates
  - Printed Reports
  - Measurement overview

Menu Builder

- Default Page Toolbar
- Data Manager Menubar
- Data Manager Toolbar
- User Menu Tree
  - End-User Menubar
    - File
    - Edit
    - Run
      - Collect Measurements
      - Reconcile
      - —
      - Stop
    - Data
    - Navigation
    - Authorization
    - Settings
    - Window
    - Help
  - Measurements
  - FlowNetwork

Figure 2.3: Overview of AIMMS tools

your model, or to discover oversights in a formulation. The **Identifier Selector** is discussed in full detail in Chapter 7

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 **Page Manager** is discussed in full detail in Section 12.1.

The Template Manager

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 **Template Manager** is discussed in full detail in Section 12.2.

The Menu Builder

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. The **Menu Builder** is discussed in full detail in Section 12.3.

2.4 Dockable windows

Support for dockable windows

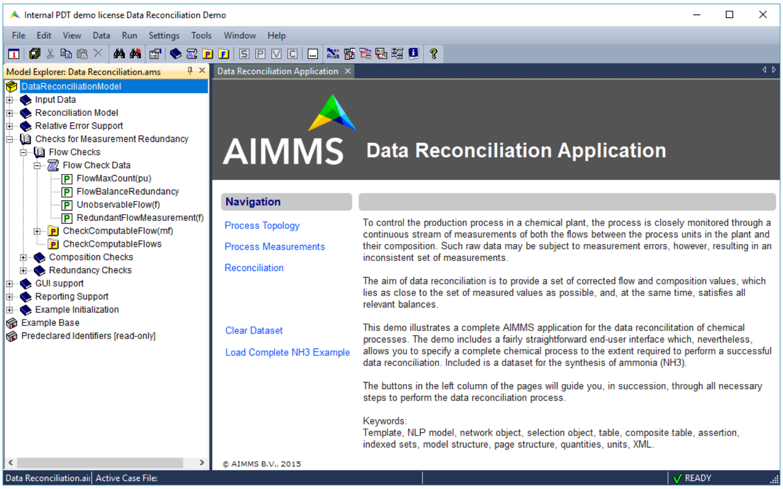
Dockable windows are an ideal means to keep frequently used tool windows in an development environment permanently visible, and are a common part of modern Integrated Development Environments (IDE) such as *Visual Studio .NET*.

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 Section 2.3 are docked at the left edge of the AIMMS window, as illustrated in Figure 2.4.



To control the production process in a chemical plant, the process is closely monitored through a continuous stream of measurements of both the flows between the process units in the plant and their composition. Such raw data may be subject to measurement errors, however, resulting in an inconsistent set of measurements.

The aim of data reconciliation is to provide a set of corrected flow and composition values, which lies as close to the set of measured values as possible, and, at the same time, satisfies all relevant balances.

This demo illustrates a complete AIMMS application for the data reconcilitation of chemical processes. The demo includes a fairly straightforward end-user interface which, nevertheless, allows you to specify a complete chemical process to the extent required to perform a successful data reconciliation. Included is a dataset for the synthesis of ammonia (NH3).

The buttons in the left column of the pages will guide you, in succession, through all necessary steps to perform the data reconciliation process.

Keywords:
Template, NLP model, network object, selection object, table, composite table, assertion, indexed sets, model structure, page structure, quantities, units, XML.

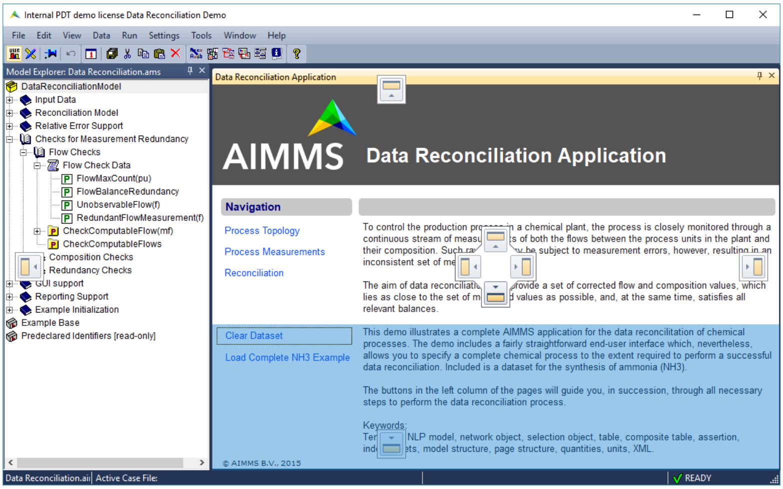
Figure 2.4: Example of a **Model Explorer** docked at the left edge

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 2.5) 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.



To control the production pr in a chemical plant, the process is closely monitored through a continuous stream of measu ts of both the flows between the process units in the plant and their composition. Such ra y be subject to measurement errors, however, resulting in an inconsistent set of me

The aim of data reconciliati provide a set of corrected flow and composition values, which lies as close to the set of m d values as possible, and, at the same time, satisfies all relevant balances.

This demo illustrates a complete AIMMS application for the data reconcilitation of chemical processes. The demo includes a fairly straightforward end-user interface which, nevertheless, allows you to specify a complete chemical process to the extent required to perform a successful data reconciliation. Included is a dataset for the synthesis of ammonia (NH3).

The buttons in the left column of the pages will guide you, in succession, through all necessary steps to perform the data reconciliation process.

Keywords:
Ter NLP model, network object, selection object, table, composite table, assertion, ind ets, model structure, page structure, quantities, units, XML.

Figure 2.5: 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 Figure 2.4 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 (see Chapter 13). 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.

2.5 Additional files related to an AIMMS project

Project-related files

In addition to the AIMMS project folders and files associated discussed in Section 2.2, 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 Section 5.3

. . . 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 Chapter 16), 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.

The user database

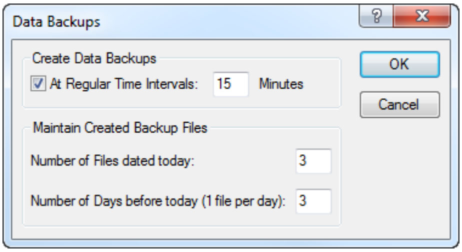
With every end-user project created with Aimms, you can associate an end-user database containing the names and passwords of all end-users entitled to run the project or view its results. Such end-user information is stored in an encrypted format in a user database file (with a `.usr` extension). You can use a single user database file with more than one project.

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 Figure 2.6. 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



Data Backups

Create Data Backups

☑ At Regular Time Intervals: 15 Minutes

Maintain Created Backup Files

Number of Files dated today: 3

Number of Days before today (1 file per day): 3

Figure 2.6: The **Data Backup** dialog box

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.

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.

2.5.1 Project user files

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 Figure 2.7. 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 Figure 2.7 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 Section 5.4). 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.

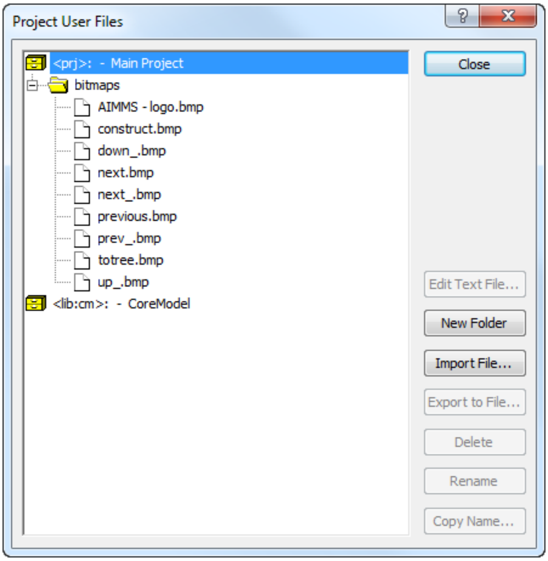


Figure 2.7: The **Project User Files** dialog box

User files in library projects

When your project, next to the main project file, also includes a number of library project files (see Section 3.1), 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 Figure 2.7, 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.

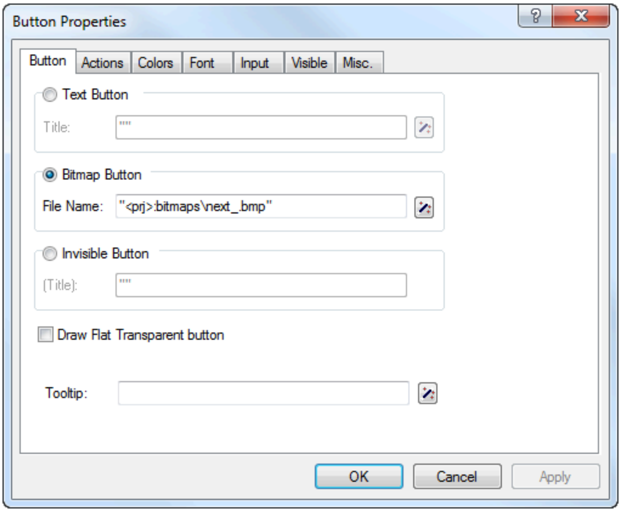
Referencing user files in library projects

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 Figure 2.7, 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. Figure 2.8 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 Figure 2.8), 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 Figure 2.7.



Button Properties

○ Text Button
Title: ""

● Bitmap Button
File Name: "<prj>:bitmaps\next_.bmp"

○ Invisible Button
(Title): ""

□ Draw Flat Transparent button

Tooltip:

Figure 2.8: Bitmap button referencing a project user file

2.6 AIMMS licensing

AIMMS licensing

AIMMS offers the following two types of licenses:

- single-user licenses, and
- network licenses.

Each of these two types of licenses are protected in a different manner.

Single-user license protection

Single-user licenses can be used by a single user on a single computer. To enforce the single-user character, AIMMS requires that single-user licenses be protected by a nodelock file, which must be activated to match the hardware characteristic of your computer.

Nodelocks

Nodelock files are stored on the harddisk of your computer, and are, therefore, much less vulnerable to loss. Only if you computer is stolen, or in case of a harddisk crash, you must contact AIMMS before being able to activate your nodelock on a replacement computer. In addition, you need access to the internet to activate or deactivate a nodelock.

Network licenses

If you have ordered an AIMMS network license, no license protection needs to be installed locally on your computer. Instead, you need the host name and port number of the server running the AIMMS network license server. For more information about installing the network license server itself, please refer to the documentation of the AIMMS network license server.

2.6.1 Personal and machine nodelocks

Two types of nodelocks

AIMMS offers two types of nodelocks:

- personal nodelocks, and
- machine nodelocks.

If you choose for nodelock protection you are free to choose between a personal or a machine type of nodelock. In this section you will find the characteristics of both types of nodelocks. If you are unsure which type of nodelock to choose, we recommend that you start with a personal nodelock, as you can change a personal nodelock into a machine nodelock at any time, but not the other way around.

Personal nodelock

Personal nodelocks are intended for use by a single AIMMS user, who still wishes to have the freedom to use AIMMS on multiple computers, for instance if you want to easily switch between your desktop computer at work, a notebook computer and your home computer. Personal nodelocks have the following characteristics:

- Personal nodelocks can be transferred to another computer 3 times per 24 hours. This allows you to take your AIMMS license home in the evening and back to work the next morning without any problems.
- Personal nodelocks have a limited lifetime of 60 days, and should be renewed within that period to extend the lifetime to its full 60-day period. If the nodelock is not renewed within its 60-day lifetime, this does not invalidate your AIMMS license in any way—you only have to renew your nodelock prior to being able to use your AIMMS system again. Note that the renewal process does not require any manual intervention, as AIMMS will try to automatically connect to our internet license database to renew your nodelock once every day you are using AIMMS.
- Both activation and nodelock renewal of personal nodelocks require a working connection to the internet. As a consequence, in the absence of an internet connection you can continue to work uninterrupted for

a period of 60 days, before an internet connection is required to renew your nodelock.
- With every activation or nodelock renewal AIMMS will also update your license files if new license files are available (e.g. if your system is in maintenance), and will inform you of any messages that are available for you in our database.
- Because of their volatile nature, AIMMS will replace a personal nodelock without any questions asked in case of loss of or damage to your computer.
- You can switch your personal nodelock to a machine nodelock at any time.

Machine nodelock

Machine nodelocks are intended for permanent use on a single computer. They are recommended for server applications, and can also be used for personal use if you are sure you will be using AIMMS on a single computer, or do not have internet access. Machine nodelocks have the following characteristics:

- Machine nodelocks can be transferred to a replacement computer 3 times per 365 days.
- Machine nodelocks have an unlimited lifetime (unless deactivated).
- Machine nodelocks can be either activated online if your computer is connected to the internet, or offline through the license activation area on the AIMMS website.
- License files will only be retrieved when the machine nodelock is activated, or by explicit request.
- In case of failure, AIMMS will, in principle, only replace machine nodelocks on the same computer.
- Once you have chosen for a machine nodelock, it is not possible to switch back to a personal nodelock.

Privacy

Although a personal nodelock and the software version check on the start page make a regular connection to the internet (the personal nodelock connects to a license database for nodelock renewal and the version check connects to a version database), we do respect your privacy and will not register patterns in your personal usage of the AIMMS software in any way. During activation no personal information will be transferred, only your computer name and some of its hardware characteristics. During deactivation we register the date and time of deactivation to enforce the transfer limit.

Internet connection and proxy settings

The connection to our internet license database is implemented as a web service. Thus, if you are able to browse the web, you should also have no trouble activating an AIMMS nodelock. If your computer connects to the internet through a proxy server, AIMMS by default tries to detect and use the proxy settings also used by Microsoft Internet Explorer.

Automatic configuration scripts

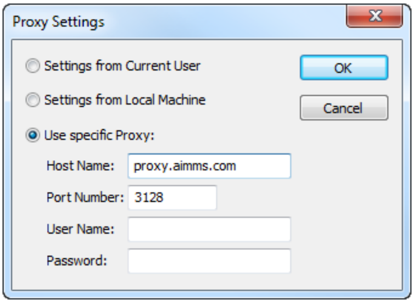
It should be noted that the use of auto-configuration scripts in determining the proxy server will fail if these use any other scripting language than Javascript. This is due to the libraries underlying the SOAP library used by AIMMS to connect to our license server. If you are in this situation, you should manually configure the proxy settings, as described below.

Manual proxy setting

If AIMMS does not succeed in automatically detecting the proxy settings that apply in your network environment, AIMMS also allows you to manually set the proxy settings during the activation process. If the online activation process does not succeed directly, AIMMS gives you the option to either continue with an offline activation process, or to manually supply the proxy settings that apply to your network environment through the dialog box illustrated in Figure 2.9. In this dialog box you can choose between

- the *Current User* settings also used by Microsoft Internet Explorer (default),
- the `Local Machine` settings which are stored in the registry, if these are available on your machine, or
- *Custom* proxy settings that you have received from your IT department.

In the latter case, you can also (optionally) provide a user name and password to authenticate with the proxy server. In most cases, however, setting these will not be necessary, and Windows authentication will be sufficient.



○ Settings from Current User

○ Settings from Local Machine

● Use specific Proxy:

Host Name: proxy.aimms.com

Port Number: 3128

User Name:

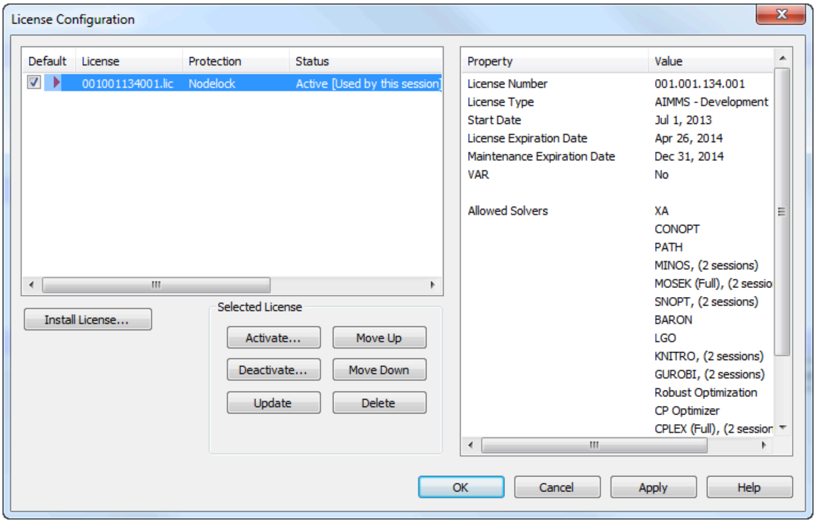
Password:

Figure 2.9: The AIMMS **Proxy Configuration** dialog box

2.6.2 Installing an AIMMS license

Managing your AIMMS licenses

When you start up AIMMS for the first time after installation, AIMMS will open the **License Configuration** dialog box illustrated in Figure 2.10. Through this dialog box you can install new AIMMS licenses and manage all AIMMS licenses that already have been installed on your computer.



Default	License	Protection	Status
☑	001001134001.lic	Nodelock	Active [Used by this session]

Property	Value
License Number	001.001.134.001
License Type	AIMMS - Development
Start Date	Jul 1, 2013
License Expiration Date	Apr 26, 2014
Maintenance Expiration Date	Dec 31, 2014
VAR	No
Allowed Solvers	XA
	CONOPT
	PATH
	MINOS, (2 sessions)
	MOSEK (Full), (2 sessio
	SNOPT, (2 sessions)
	BARON
	LGO
	KNITRO, (2 sessions)
	GUROBI, (2 sessions)
	Robust Optimization
	CP Optimizer
	CPLEX (Full), (2 session

Figure 2.10: The **License Configuration** dialog box

Installing a new license

To install a new license, press the **Install License ...** button in the **License Configuration** dialog box. This will start a wizard, that will guide you through the license installation procedure step by step. The wizard can help you to install

- existing AIMMS licenses,
- nodelocked licenses,
- network licenses,
- evaluation licenses, and
- student licenses.

To successfully complete the installation of licenses of each type, you should make sure to have the following information available.

Single-user nodelocked licenses

To install a single-user AIMMS license that is protected by a nodelock, you need the following information:

- your AIMMS license number, and
- the associated activation code that you received from AIMMS.

You have the choice to request a personal nodelock or a machine nodelock. A personal nodelock must be requested online, a machine nodelock can be requested online or offline. Refer to Section 2.6.1 for a more detailed introduction to personal and machine nodelocks.

Network licenses

To install an AIMMS network license, you need the following information from your system administrator:

- the name of the AIMMS network license server,
- the port number of the AIMMS network license server, and
- the name of the license profile to which you want to connect (optional).

Evaluation licenses

To install an AIMMS evaluation license you need the following information

- your AIMMS evaluation license number, and
- the associated activation code that you received from AIMMS when requesting an evaluation license.

You must have a working connection to the internet (not necessarily on the machine on which you installed AIMMS) to activate an evaluation license. Evaluation licenses expire 30 days after activation. Note that each evaluation license can be activated only once, and that you can only activate a single evaluation license per AIMMS release on a specific computer, regardless of the number of evaluation licenses you have requested on our web site.

Student licenses

To install an AIMMS student license you need the following information:

- your AIMMS student license number, and
- the associated activation code that you received from the university that purchased the AIMMS Educational Package.

You must have a working connection to the internet to activate a student license. Student licenses expire one month after the end of the current academic year. Student licenses can be activated multiple times.

2.6.3 Managing AIMMS licenses

Managing multiple AIMMS *licenses*

AIMMS allows you to have multiple AIMMS licenses installed on your computer. You may have multiple licenses installed, for instance, for the following reasons:

- you have requested a trial license for a new AIMMS version which you want to run next to your existing license,
- you have temporarily borrowed or hired an AIMMS license with more capabilities than your regular license,
- your system administrator has created multiple network license profiles, each of which you may want to use to run AIMMS.

In this section we will describe how you can instruct AIMMS which license to use.

Default licenses

In the **License Configuration** dialog box displayed in Figure 2.10, all AIMMS licenses installed on your machine will be displayed in the left pane of the dialog box. The license details of each license are displayed in the right pane of the dialog box. During startup AIMMS will consider all licenses in the left pane of the **License Configuration** dialog box which have the **Default** column checked, and will use the first valid license it finds starting from top to bottom. Using the **Move Up** and **Move Down** buttons you can change the order in which AIMMS will search the list.

Transferring licenses

Both personal and machine nodelocks can be transferred to other computers. Personal nodelocks can be transferred upto three times a day, allowing you to take your license with you wherever you want. Machine nodelocks can be transferred three times per year, to a computer replacing the computer on which the nodelock is currently installed. To transfer a nodelocked license, you must

- deactivate the nodelock on the currently active computer, and
- activate it on the computer to which you want to transfer the license.

You can deactivate an active nodelock using the **Deactivate** button in the **License Configuration** dialog box. Deactivation will only succeed if there is no conflict with the transfer limit for the given nodelock type. This makes sure that there will never be a problem activating a deactivated license. After successful deactivation the license will not be removed from the list but be marked as inactive. If the license is not active on any computer, you can reactivate the license through the **Activate** button.

Emergency nodelocks

In case you want to activate a nodelock on a computer, but have forgotten to deactivate the nodelock on a computer to which you currently have no access, AIMMS allows you, as a courtesy, to request an emergency nodelock 3 times per 365 days. Emergency nodelocks have a lifetime of 7 days, and during this time you can arrange for someone to deactivate the license on the computer containing the active nodelock. During the activation sequence, AIMMS will automatically ask whether you would like to receive an emergency nodelock when it discovers that the license is active on another computer.

2.6.4 Location of license files

Location of license files

AIMMS keeps its license and configuration files in the folder

```
AIMMS
```

of the common application area of your computer. On Windows 7, 8 and 10, this folder is located under `C:\ProgramData`. The `AIMMS` folder contains five subfolders

- `Config`, containing the license and solver configuration files,

- `Licenses`, containing all license files,
- `Nodelocks`, containing all nodelock files installed on your computer,
- `ApplicationKeys`, containing any private/public key pairs that you have generated, and
- `AnyUser`, containing the license configuration files for all users on your computer (see below).

The AIMMS installation makes sure that these subfolders are writable for everyone, allowing you to install and uninstall licenses on your computer.

Do not move nodelock files

To prevent tampering with nodelocked licenses, AIMMS keeps track of the location of the nodelock files associated with a license. You should, therefore, not manually move or copy the AIMMS nodelock files as this may invalidate your nodelock.

User specific configuration

You can specify whether the license and solver configuration that AIMMS uses is the same for any user of the machine, or different for each individual user. To prevent problems when running AIMMS as part of a computer service, AIMMS will by default use the same configuration for any user. To modify this behavior, you should edit the file `UserDistinction.cfg` in the common `Config` folder. In the file `UserDistinction.cfg.default`, straightforward directions are given on how to edit it. If no `UserDistinction.cfg` file exists, AIMMS will use the `UserDistinction.cfg.default` file instead. You can use this file as a base for setting up your own configuration.

Any User

In the scenario where all users of the same pc use the same license and solver configuration, the configurations that are modified by a user are stored in the `AnyUser` folder of the `Common` folder.

Current User

In the scenario where each specific user of the pc has its own license and solver configuration, the configurations that are modified by a user are stored in the local application data folder. On Windows 7, 8 and 10 it is:

```
C:\Users\<UserName>\AppData\Roaming\AIMMS
```

If you have already used AIMMS 3 on your machine, these folders don't end in `AIMMS`, but (still) in `Paragon Decision Technology`.

Accessing configuration files

When AIMMS needs to read the current configuration, it will first look in the `(any)user` folder as specified by the aforementioned `UserDistinction.cfg` file; if it cannot be found there, it will try to read the configuration from the common application data folder. When saving a modified configuration, AIMMS always writes to the `(any)user` folder.

Version dependent configurations

Inside the `Config` folder of the `(any)user` folder, each major AIMMS version (3.6, 3.7, 3.8, ..., 4.0 etc.) will create its own specific subfolder when it needs to write a configuration file. During an attempt to read, AIMMS will first look for the specific file in the folder that matches its own major version number, and otherwise it will subsequently search through the folders of previous versions. In other words, when you upgrade to a new AIMMS version, initially your configuration will be the same as the one you were using for the previous AIMMS version, but if you change something in the configuration, this will only affect the configuration of the AIMMS version you are working with.

Network License Client Files

If you are using an AIMMS Network License, then your local machine does not need to have any license file installed. The only required file is the license configuration file, that contains the info of where the License Server is located on your LAN. When logging on to the License Server, the licensing info is sent directly from the server to the running AIMMS session, except for some secondary license related files:

- the `.SLV` file (containing the default solver configuration), and
- the `.CPX` file (the *CPLEX* license file).

These secondary license files are temporarily copied to the folder `NetworkCache` which is located in the `(any)user` folder as described above.

Project dependent configuration

You can specify a project dependent solver configuration by placing a solver configuration file with the name `'solvers.slv'` in the project directory. AIMMS will first look for this file and if it cannot find it will look for other solver configuration files. See the AIMMS Help for more information.

Chapter 3

Collaborative Project Development

This chapter

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.

... 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.

3.1 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

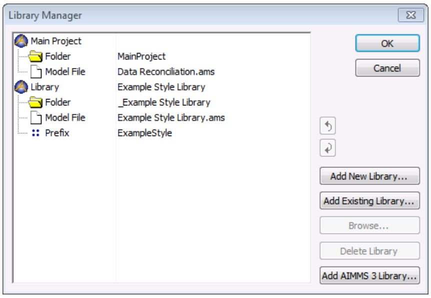
Besides enabling multiple developers to work in a single project, library projects can also be used to define a common collection of templates that define the look-and-feel of multiple projects. In this manner, you change the look-and-feel of multiple applications just by changing the templates in the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

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 Figure 3.1. You can open the library manager through the **File-Library Manager...** menu.



Main Project	
Folder	MainProject
Model File	Data Reconciliation.ams
Library	Example Style Library
Folder	_Example Style Library
Model File	Example Style Library.ams
Prefix	ExampleStyle

Figure 3.1: The **AIMMS Library Manager**

Using the library manager AIMMS allows you to

- create new libraries,
- add existing libraries to your project,
- edit library properties, and
- remove libraries from your project.

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.

3.2 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 page 34. 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 Section 35.5 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.

Part II

Creating and Managing a Model

Chapter 4

The Model Explorer

This chapter

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.

4.1 What is the Model Explorer?

Support for large models

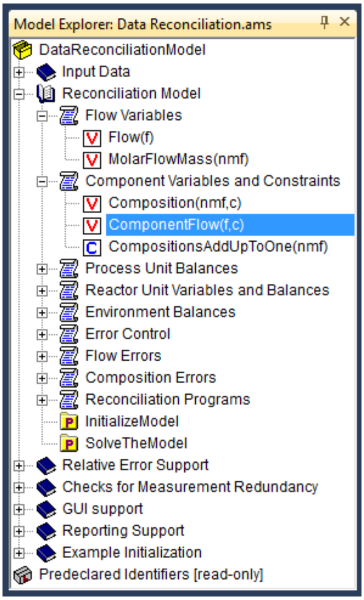
Decision making commonly requires access to massive amounts of information on which to base the decision making process. As a result, professional decision support systems are usually very complex programs with hundreds of (indexed) identifiers to store all the data that are relevant to the decision 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 Figure 4.1.
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 Figure 4.1, 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.



Model Explorer: Data Reconciliation.ams

- DataReconciliationModel
  - Input Data
  - Reconciliation Model
    - Flow Variables
      - Flow(f)
      - MolarFlowMass(nmf)
    - Component Variables and Constraints
      - Composition(nmf,c)
      - ComponentFlow(f,c)
      - CompositionsAddUpToOne(nmf)
    - Process Unit Balances
    - Reactor Unit Variables and Balances
    - Environment Balances
    - Error Control
    - Flow Errors
    - Composition Errors
    - Reconciliation Programs
    - InitializeModel
    - SolveTheModel
  - Relative Error Support
  - Checks for Measurement Redundancy
  - GUI support
  - Reporting Support
  - Example Initialization
- Predeclared Identifiers [read-only]

Figure 4.1: 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 Section 3.1). 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 Chapter 35 of the Language Reference.

Aimms *library*

For your convenience, Aimms always includes a single, read-only library module called `Predeclared Identifiers` (displayed in Figure 4.1), 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 [S][P][V][C][...] (the last button opens a selection list of all available identifier types). Identifier declarations are explained in full detail in Chapter 5.

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

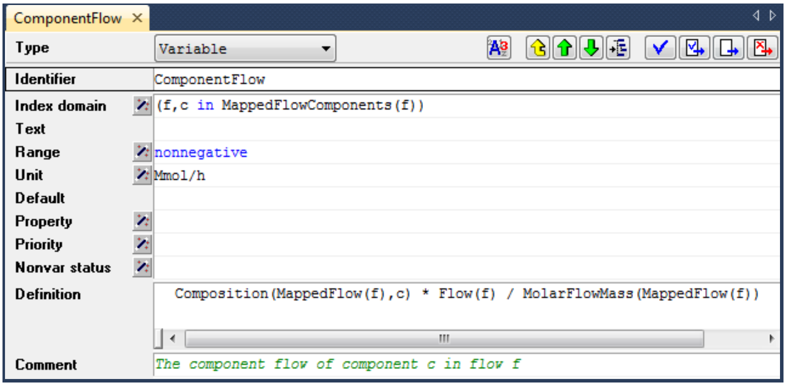
Another frequently occurring node type is the declaration of a procedure or a function. Such a procedure or function node contains the data retrieval statements, computations, and algorithms that make up the procedural execution of your modeling application. Procedures and functions are represented by folder icons, P and F, 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 Chapter 6.

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 Chapter 6.

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 Figure 4.2. 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.



ComponentFlow ×

Type	Variable
Identifier	ComponentFlow
Index domain	(f,c in MappedFlowComponents(f))
Text	
Range	nonnegative
Unit	Mmol/h
Default	
Property	
Priority	
Nonvar status	
Definition	Composition(MappedFlow(f),c) * Flow(f) / MolarFlowMass(MappedFlow(f))
Comment	The component flow of component c in flow f

Figure 4.2: 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 Figure 4.2. 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 Section 4.2.

4.2 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 Section 4.1), 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

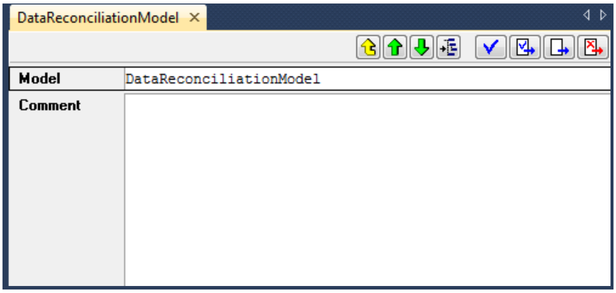
When you feel that particular groups of declarations, procedures and functions belong together in a logical manner, you are encouraged to create a new structuring section with a descriptive name within the model tree, and store the associated model components within it. When your application grows in 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`" (AIMMS model source) 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 Figure 4.3. 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


DataReconciliationModel

Model	DataReconciliationModel
Comment	

Figure 4.3: Attribute form of a section node

a consequence, AIMMS will automatically read the contents of the book section from that file during every subsequent session.

Protecting source files

Section 19.1 explains how you can further protect such a `.ams` file by encrypting its contents, allowing you to ship it to your customers as an end-user only module.

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.

4.2.1 Working with modules and libraries

Name clashes

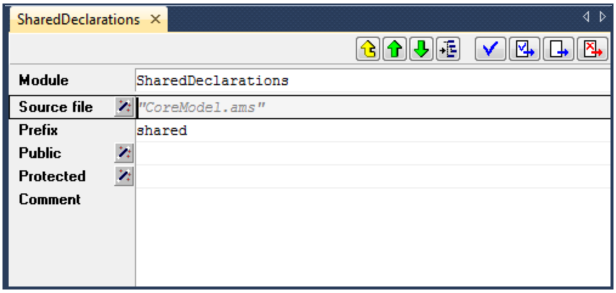
When you import the contents of a book section node into your model, you may find that particular identifier names in that book section already have been declared in the remainder of your model. If such a name clash occurs, AIMMS will refuse to import the specified `.ams` file into your model, and present 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 Chapter 35 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 Section 4.3. For each module you must specify a module prefix through which you can access the identifiers stored in the module. Figure 4.4 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`.



Module	SharedDeclarations
Source file	"CoreModel.ams"
Prefix	shared
Public	
Protected	
Comment	

Figure 4.4: The attributes of a *Module* node

AIMMS *system modules*

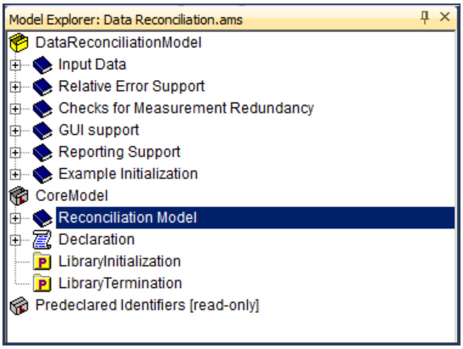
AIMMS uses modules to implement those parts of its functionality that can be best expressed in the AIMMS language itself. The available AIMMS system modules include

- 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 Section 3.1. For each library included in your project, AIMMS creates a separate library module node at the root the **Model Explorer**, as illustrated in Figure 4.5. When creating a new library the associated library module will initially be empty. In the library module of Figure 4.5, one section from the original model tree in Figure 4.1 has already been moved into the newly created library.


Model Explorer: Data Reconciliation.ams
DataReconciliationModel
- Input Data
- Relative Error Support
- Checks for Measurement Redundancy
- GUI support
- Reporting Support
- Example Initialization

CoreModel
- Reconciliation Model
- Declaration
- LibraryInitialization
- LibraryTermination

Predeclared Identifiers [read-only]

Figure 4.5: 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 Section 5.2.1, 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 Section 25.1 of the Language Reference.

4.3 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^{TM} 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 P F … or S P V C … 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 Figure 4.6 appears from which you have to select a node type.

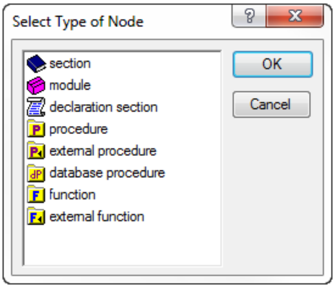


Figure 4.6: Dialog box for selecting a node type

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 Figure 4.6), 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 Chapter 7.
- 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.

Chapter 5

Identifier Declarations

This chapter

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.

5.1 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 Section 4.3. 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 Figure 5.1.

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.

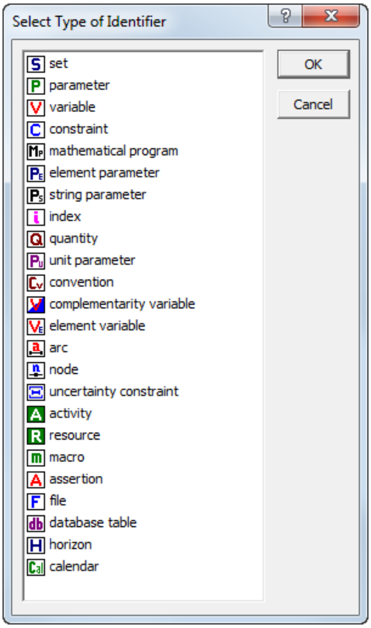


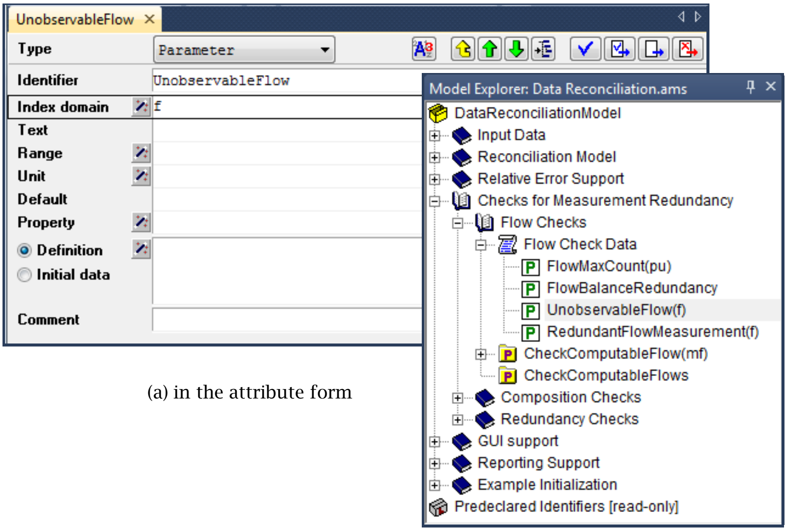
Figure 5.1: Choosing an identifier type

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 Section 5.2), 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. Figure 5.2 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.



UnobservableFlow

Type: Parameter

Identifier: UnobservableFlow

Index domain: f

(a) in the attribute form

(b) in the model explorer

Figure 5.2: 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.

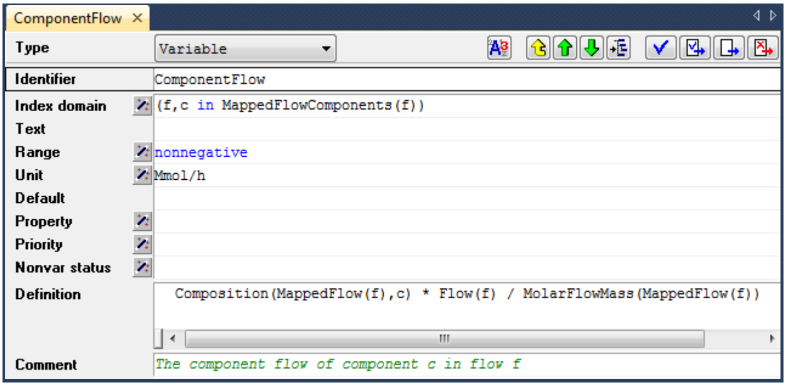
5.2 Identifier attributes

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. Figure 5.3 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.



Type	Variable
Identifier	ComponentFlow
Index domain	(f,c in MappedFlowComponents(f))
Text	
Range	nonnegative
Unit	Mmol/h
Default	
Property	
Priority	
Nonvar status	
Definition	Composition(MappedFlow(f),c) * Flow(f) / MolarFlowMass(MappedFlow(f))
Comment	The component flow of component c in flow f

Figure 5.3: Identifier attributes

Default values

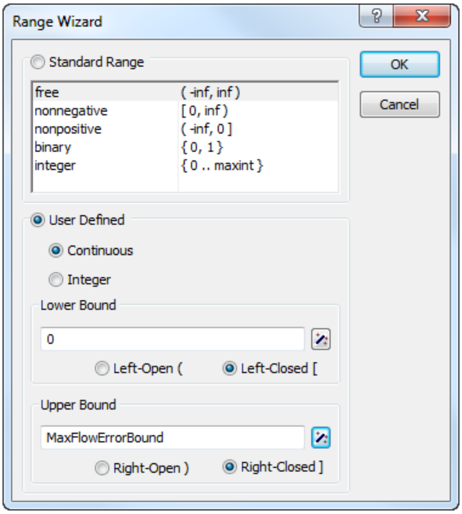
You do not need to enter values for all the attributes in an attribute window. In fact, most of the attributes are optional, or have a default value (which is not shown). You only have to enter an attribute value when you want to alter the behavior of the identifier, or when you want to provide a value 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 Figure 5.4. 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.



Range Wizard

○ Standard Range

free	(-inf, inf)
nonnegative	[0, inf)
nonpositive	(-inf, 0]
binary	{ 0, 1 }
integer	{ 0 .. maxint }

● User Defined

● Continuous

○ Integer

Lower Bound

0

○ Left-Open (● Left-Closed [

Upper Bound

MaxFlowErrorBound

○ Right-Open) ● Right-Closed]

Figure 5.4: 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 Figure 5.5. 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.

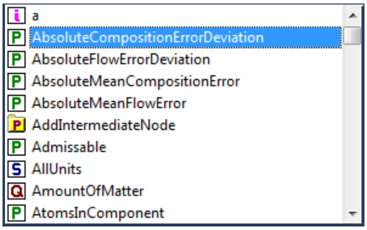


Figure 5.5: Name completion

. . . 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).

5.2.1 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 Section 5.4).

Context help 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.

5.3 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.

- **Check and commit** ✓. This command checks the current values of the attributes for syntax errors, and if there are no errors the new values are applied to the model.
- **Check, commit and close** ☑. Same as check and commit, but if there are no errors it also closes the current form. Since this is the most frequently used action, you can also invoke it by pressing **Ctrl-Enter**.
- **Commit and close** ☐. This command does *not* check the current values, but simply applies them to the model and then closes the form. The changes will be checked later, when the entire model is checked or when you re-open and check the form yourself.
- **Discard** ☒. If you do not want to keep any of the changes you made in the attribute form, you can discard them using the **Discard** button.

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 Section 2.5), 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

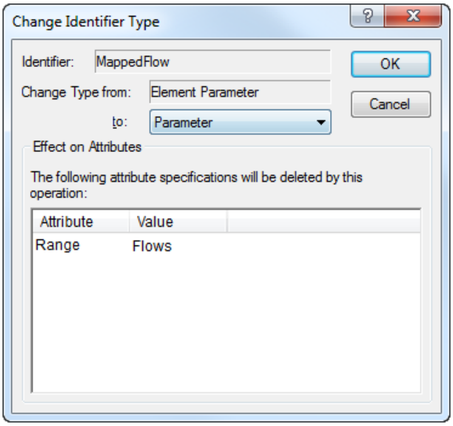
Problems arise when you want to change the index domain of an identifier, or remove an identifier, while it is still referenced somewhere in your application. Such changes are called *structural*, and are likely to cause errors in pages and cases. In general, these errors cannot be recovered automatically. 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 Figure 5.6, which lists all the attributes of the identifier that are not compatible with the newly selected type. 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.


Change Identifier Type

Identifier: MappedFlow

Change Type from: Element Parameter

to: Parameter

Effect on Attributes

The following attribute specifications will be deleted by this operation:

Attribute	Value
Range	Flows

Figure 5.6: The **Change Identifier Type** dialog box

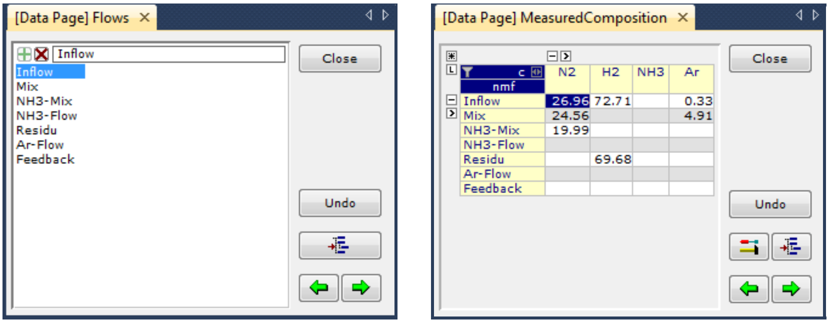
5.4 Viewing and modifying identifier data

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 Section 8.1) 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, Figure 5.3), AIMMS will pop up one of the *data pages* as illustrated in Figure 5.7. 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.



Inflow

Inflow
Mix
NH3-Mix
NH3-Flow
Residu
Ar-Flow
Feedback

c \ nmf	N2	H2	NH3	Ar
Inflow	26.96	72.71		0.33
Mix	24.56			4.91
NH3-Mix	19.99			
NH3-Flow				
Residu		69.68		
Ar-Flow				
Feedback				

Figure 5.7: Data pages of a set and a 2-dimensional parameter

Data pages for variables and constraints

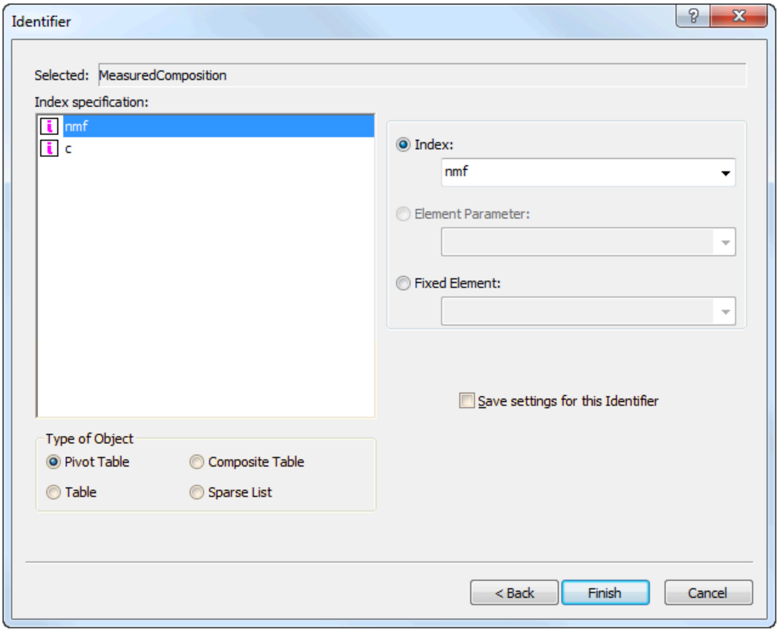
For variables (and similarly for constraints), AIMMS will display a pivot table containing all the indices from the index domain of the variable plus one additional dimension containing all the suffices of the variable that contain relevant information regarding the solution of the variable. Depending 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 Figure 5.8. 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* (see also Section 10.4), 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.



Identifier

Selected: MeasuredComposition

Index specification:

- nmf
- c

● Index: nmf

○ Element Parameter:

○ Fixed Element:

□ Save settings for this Identifier

Type of Object

● Pivot Table ○ Composite Table

○ Table ○ Sparse List

Figure 5.8: Selecting a data page type

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 Figure 5.8. 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** (see also Section 12.1). 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 Section 8.1), 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.

Chapter 6

Procedures and Functions

This chapter

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.

6.1 Creating procedures and functions

Procedures and functions

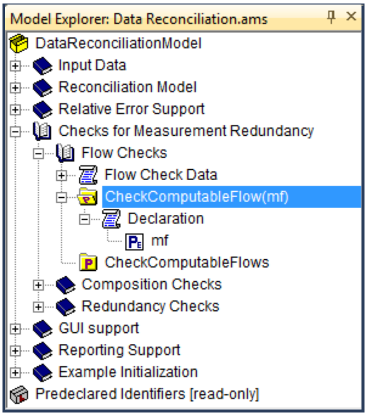
Procedures and functions are the main means of executing the sequential tasks in your model that cannot be expressed by simple functional relationships in the form of identifier definitions. Such tasks include importing or exporting your model's data from or to an external data source, the execution 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. Figure 6.1 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 Section 6.2. Whether or not the arguments are fully visible in the tree is configurable using the **View** menu.



Model Explorer: Data Reconciliation.ams

- DataReconciliationModel
  - Input Data
  - Reconciliation Model
  - Relative Error Support
  - Checks for Measurement Redundancy
    - Flow Checks
      - Flow Check Data
      - CheckComputableFlow(mf)
        - Declaration
          - mf
      - CheckComputableFlows
    - Composition Checks
    - Redundancy Checks
  - GUI support
  - Reporting Support
  - Example Initialization
- Predeclared Identifiers [read-only]

Figure 6.1: Example of a procedure node

Procedure and function attributes

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. An example of the attribute window of a procedure node within the model tree is illustrated in Figure 6.2. The contents of the Body attribute is application-specific, and is irrelevant to a further understanding of the material in this section.

[0.49]bordercolorproc-attr-new

Figure 6.2: Example of procedure attributes

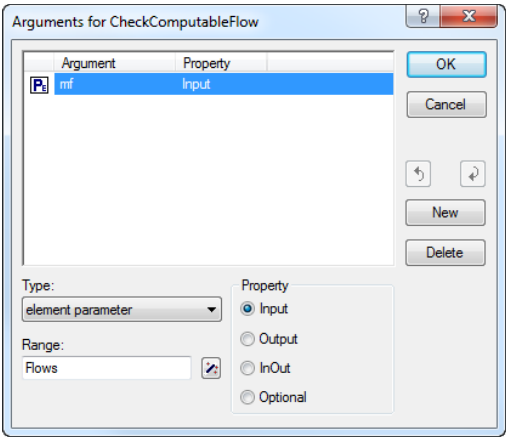
Specifying function domain and range

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.

6.2 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 Figure 6.3 which helps you add the appropriate declaration quickly and easily. 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 Figure 6.1.



Arguments for CheckComputableFlow

Argument	Property
mf	Input

Type: element parameter

Range: Flows

Property

● Input

○ Output

○ InOut

○ Optional

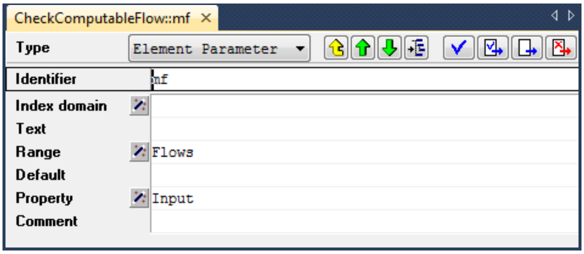
Figure 6.3: **Argument Declaration** dialog box

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, Figure 6.4 shows the attribute form of the single argument `mf` of the procedure `CheckComputableFlow` added in Figure 6.3. In the dialog box of Figure 6.3 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.



Type	Element Parameter
Identifier	mf
Index domain	
Text	
Range	Flows
Default	
Property	Input
Comment	

Figure 6.4: 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.

6.3 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`

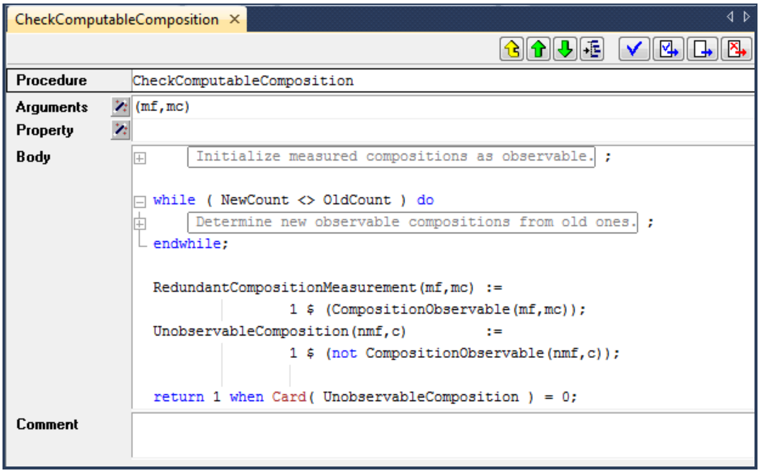
which perform the actual task or computation for which the procedure or function is intended. The precise syntax of all execution statements is discussed in detail in the AIMMS Language Reference book.

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 Figure 6.5.



Procedure: CheckComputableComposition

Arguments: (mf,mc)

Property:

Body:

```
⊞ Initialize measured compositions as observable. ;

⊟ while ( NewCount <> OldCount ) do
⊞    Determine new observable compositions from old ones. ;
  endwhile;

  RedundantCompositionMeasurement(mf,mc)  :=
                  1 $ (CompositionObservable(mf,mc));
  UnobservableComposition(nmf,c)          :=
                  1 $ (not CompositionObservable(nmf,c));

  return 1 when Card( UnobservableComposition ) = 0;
```

Comment:

Figure 6.5: 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 Sections 4.3 and 5.2.

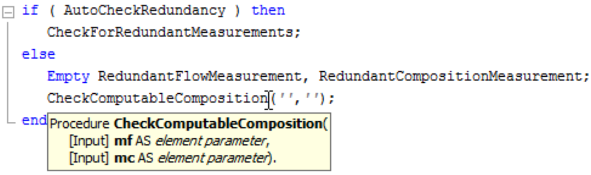
Viewing identifier details

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 Section 4.3),
- by clicking on a particular identifier reference in the body, you can open its attributes form through the **Attributes** item in the right-mouse popup 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 Section 5.4).

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 Figure 6.6. 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.



```
if ( AutoCheckRedundancy ) then
   CheckForRedundantMeasurements;
else
   Empty RedundantFlowMeasurement, RedundantCompositionMeasurement;
   CheckComputableComposition('','');
end
```

Procedure **CheckComputableComposition**(
  [Input] **mf** AS *element parameter*,
  [Input] **mc** AS *element parameter*).

Figure 6.6: Prototype info of a procedure

6.4 Syntax checking, compilation and execution

Performing a syntax check

Using either **Check and commit** or **Check, commit and close** as discussed in Section 5.3 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.

Chapter 7

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.

This chapter

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.

7.1 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

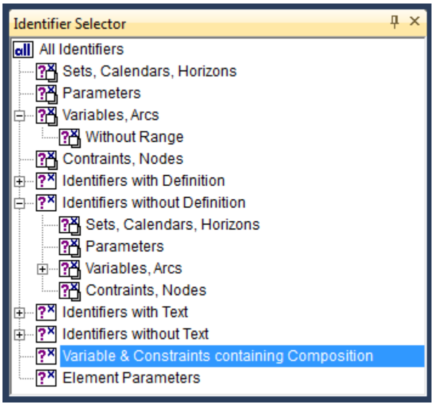
If you need a selection only once, then you can create it directly in the **Model Explorer** by

- 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. Figure 7.1 illustrates an example selector tree.



Identifier Selector

- All Identifiers
  - Sets, Calendars, Horizons
  - Parameters
  - Variables, Arcs
    - Without Range
  - Contraints, Nodes
  - Identifiers with Definition
  - Identifiers without Definition
    - Sets, Calendars, Horizons
    - Parameters
    - Variables, Arcs
    - Contraints, Nodes
  - Identifiers with Text
  - Identifiers without Text
  - Variable & Constraints containing Composition
  - Element Parameters

Figure 7.1: 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 (≤, = or ≥). This selector can only be used in combination with the Math Program Inspector.

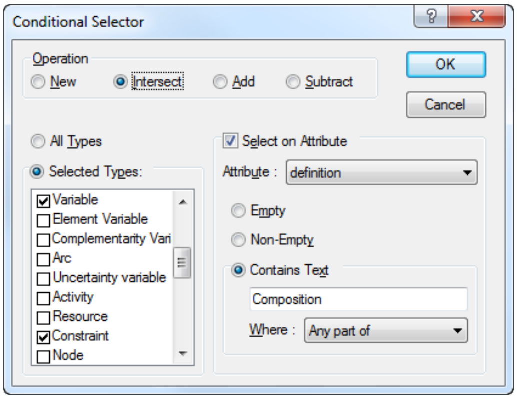
While the above four selectors allow you to define selections based on a symbolic criteria, the four types of identifier selectors below allow you to specify selections based on individual criteria. The main purpose of these selectors is to define selections that can be used in the Math Program Inspector (see Chapter 9).

- 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 Figure 7.2. In it, you can select (by double clicking) one or more identifier types that you want to be part of the selection


Conditional Selector

Operation
○ New ● Intersect ○ Add ○ Subtract

○ All Types
● Selected Types:
☑ Variable
□ Element Variable
□ Complementarity Vari
□ Arc
□ Uncertainty variable
□ Activity
□ Resource
☑ Constraint
□ Node

☑ Select on Attribute
Attribute : definition
○ Empty
○ Non-Empty
● Contains Text
Composition
Where : Any part of

Figure 7.2: The **Conditional Selector** dialog box

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.

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 Chapter 9), 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.

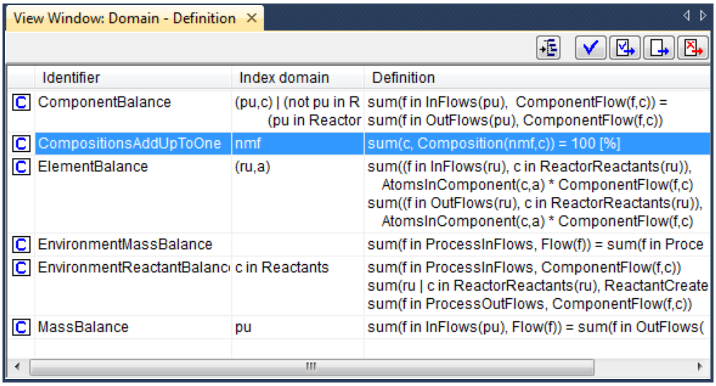
7.2 Viewing identifier selections

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 Figure 7.3.



	Identifier	Index domain	Definition
C	ComponentBalance	(pu,c) | (not pu in R (pu in Reactor	sum(f in InFlows(pu), ComponentFlow(f,c)) = sum(f in OutFlows(pu), ComponentFlow(f,c))
C	CompositionsAddUpToOne	nmf	sum(c, Composition(nmf,c)) = 100 [%]
C	ElementBalance	(ru,a)	sum((f in InFlows(ru), c in ReactorReactants(ru)), AtomsInComponent(c,a) * ComponentFlow(f,c) sum((f in OutFlows(ru), c in ReactorReactants(ru)), AtomsInComponent(c,a) * ComponentFlow(f,c)
C	EnvironmentMassBalance		sum(f in ProcessInFlows, Flow(f)) = sum(f in Proce

C	EnvironmentReactantBalanc	c in Reactants	sum(f in ProcessInFlows, ComponentFlow(f,c))
			sum(ru | c in ReactorReactants(ru), ReactantCreate
			sum(f in ProcessOutFlows, ComponentFlow(f,c))

C	MassBalance	pu	sum(f in InFlows(pu), Flow(f)) = sum(f in OutFlows(

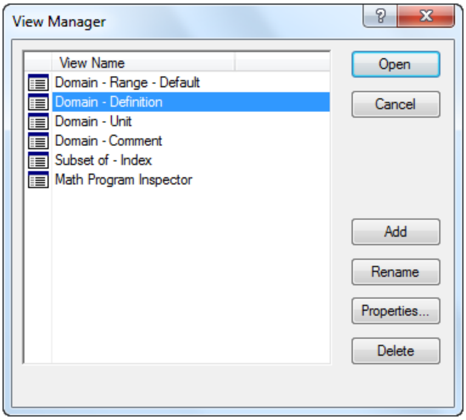
Figure 7.3: Example of a **View** window

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 Section 5.3).

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 Figure 7.4. 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.

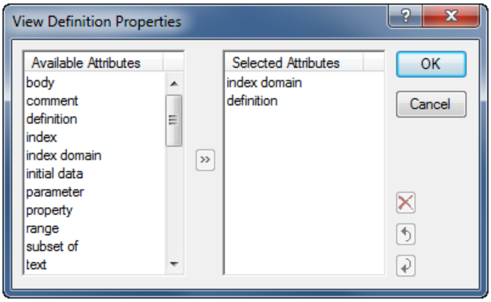


View Name
Domain - Range - Default
Domain - Definition
Domain - Unit
Domain - Comment
Subset of - Index
Math Program Inspector

Figure 7.4: The **View Manager** dialog box

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 Figure 7.5. With this dialog box you can



Available Attributes
body
comment
definition
index
index domain
initial data
parameter
property
range
subset of
text

Selected Attributes
index domain
definition

Figure 7.5: **View Definition Properties** dialog box

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.

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 Figure 7.4, 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.

Chapter 8

Debugging and Profiling an AIMMS Model

This chapter

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.

8.1 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 Chapter 9. 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 **Debug-**

ger menu to the system menu bar, and, in addition, add the **Debugger** toolbar illustrated in Figure 8.1 to the toolbar area. You can stop the AIMMS debugger

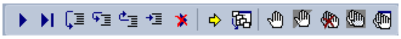


Figure 8.1: The **Debugger** toolbar

through the **Debugger-Exit Debugger** menu.

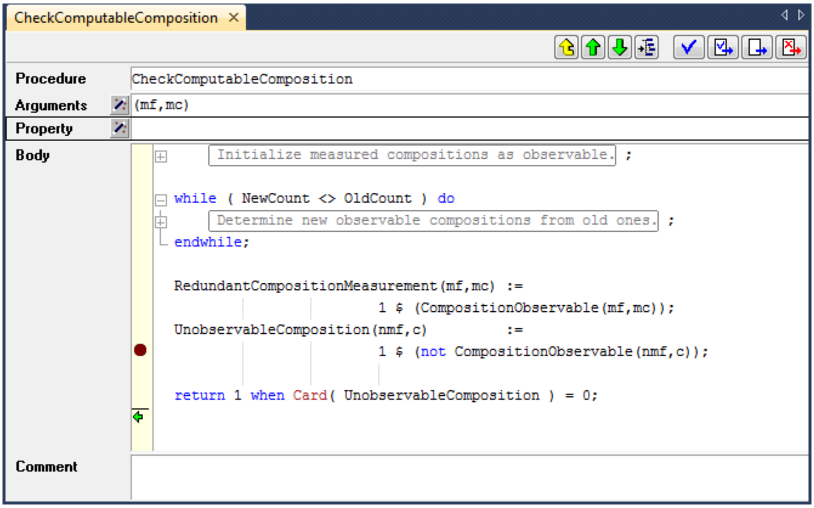
Debugger functionality

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 Section 5.4)

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 Figure 8.2.



Procedure: CheckComputableComposition

Arguments: (mf,mc)

Property:

Body:

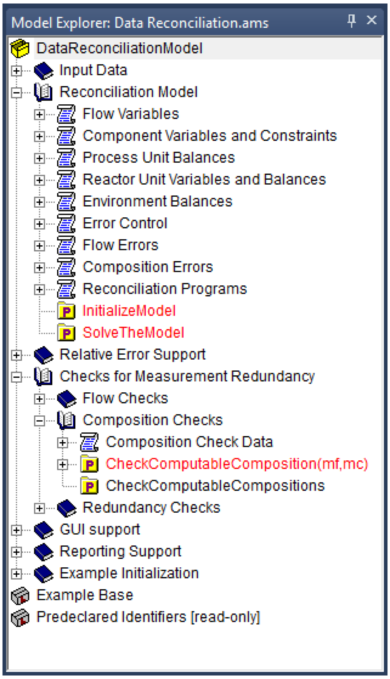
```
Initialize measured compositions as observable. ;

while ( NewCount <> OldCount ) do
    Determine new observable compositions from old ones. ;
endwhile;

RedundantCompositionMeasurement(mf,mc)  :=
                       1 $ (CompositionObservable(mf,mc));
UnobservableComposition(nmf,c)          :=
                       1 $ (not CompositionObservable(nmf,c));

return 1 when Card( UnobservableComposition ) = 0;
```

Comment:

Figure 8.2: 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 Figure 8.3.



Model Explorer: Data Reconciliation.ams

- DataReconciliationModel
- Input Data
- Reconciliation Model
  - Flow Variables
  - Component Variables and Constraints
  - Process Unit Balances
  - Reactor Unit Variables and Balances
  - Environment Balances
  - Error Control
  - Flow Errors
  - Composition Errors
  - Reconciliation Programs
  - InitializeModel
  - SolveTheModel
- Relative Error Support
- Checks for Measurement Redundancy
  - Flow Checks
  - Composition Checks
    - Composition Check Data
    - CheckComputableComposition(mf,mc)
    - CheckComputableCompositions
  - Redundancy Checks
- GUI support
- Reporting Support
- Example Initialization
- Example Base
- Predeclared Identifiers [read-only]

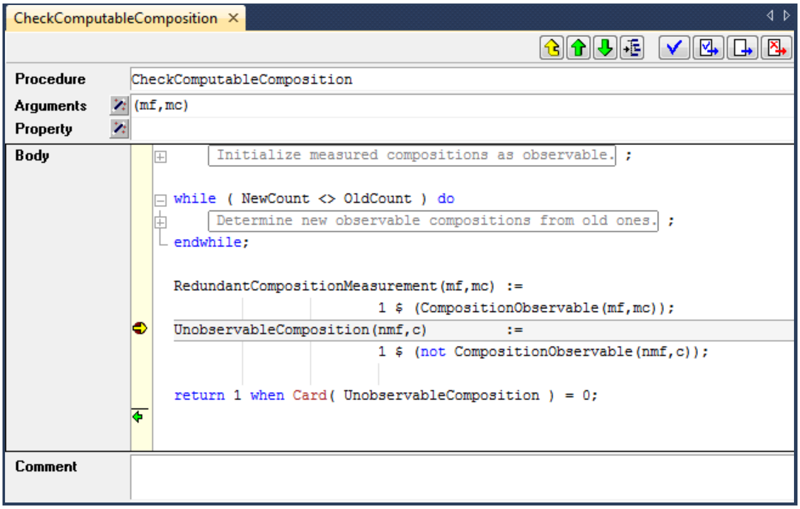
Figure 8.3: Viewing procedures with breakpoints in the **Model Explorer**

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 Figure 8.4.



Procedure: CheckComputableComposition

Arguments: (mf,mc)

Property:

Body:

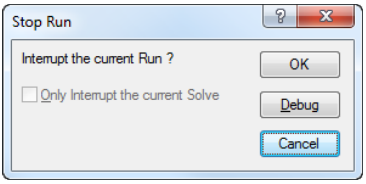
```
Initialize measured compositions as observable. ;

while ( NewCount <> OldCount ) do
    Determine new observable compositions from old ones. ;
endwhile;

RedundantCompositionMeasurement(mf,mc) :=
                            1 $ (CompositionObservable(mf,mc));
UnobservableComposition(nmf,c)          :=
                            1 $ (not CompositionObservable(nmf,c));

return 1 when Card( UnobservableComposition ) = 0;
```

Comment:

Figure 8.4: 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 Figure 8.5 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.



Stop Run

Interrupt the current Run ?

□ Only Interrupt the current Solve

Figure 8.5: 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*.

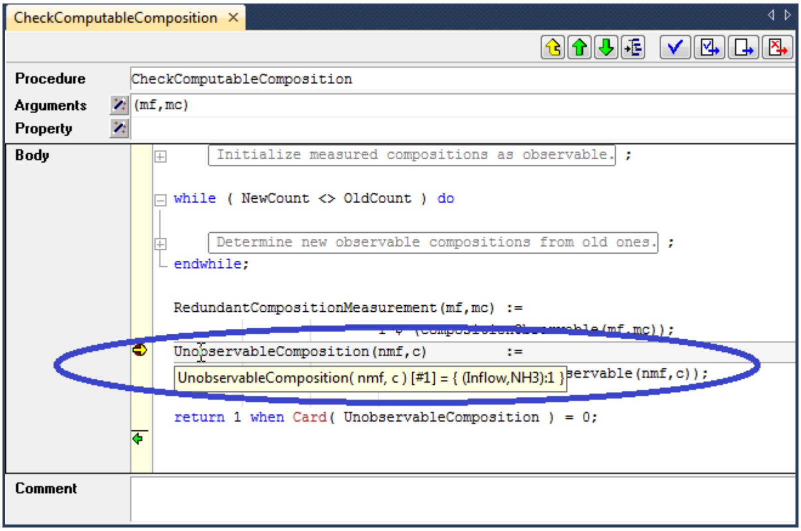
In addition, AIMMS offers some methods to continue or halt the execution:

- 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 Figure 8.6. 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.



```
Procedure   CheckComputableComposition
Arguments   (mf,mc)
Property
Body
    Initialize measured compositions as observable. ;

    while ( NewCount <> OldCount ) do

        Determine new observable compositions from old ones. ;
    endwhile;

    RedundantCompositionMeasurement(mf,mc) :=
                    1 $ (CompositionObservable(mf,mc));
    UnobservableComposition(nmf,c)         :=
    UnobservableComposition( nmf, c ) [#1] = { (Inflow,NH3):1 }ervable(nmf,c));

    return 1 when Card( UnobservableComposition ) = 0;
Comment
```

Figure 8.6: Observing the current data of an identifier through a tooltip

Detailed identifier data

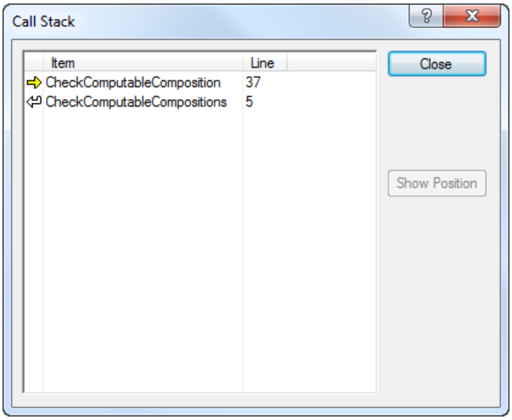
If you need to examine the effect of a statement on the data of a particular identifier in more detail, you can simply open a **Data Page**, as described in Section 5.4, or observe the effect on ordinary end-user pages. Within a debugger session, AIMMS supports data pages for both global and local identifiers, 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 Figure 8.7. 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.

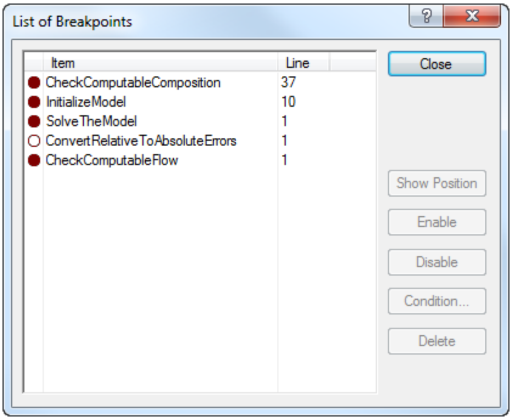


Item	Line
⇨ CheckComputableComposition	37
↩ CheckComputableCompositions	5

Figure 8.7: The **Call Stack** dialog box

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 Figure 8.8. 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



Item	Line
● CheckComputableComposition	37
● InitializeModel	10
● SolveTheModel	1
○ ConvertRelativeToAbsoluteErrors	1
● CheckComputableFlow	1

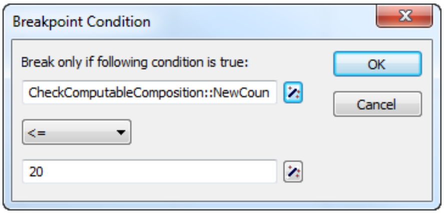
Figure 8.8: The **List of Breakpoints** dialog box

- 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 Figure 8.9. 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.



Breakpoint Condition

Break only if following condition is true:

CheckComputableComposition::NewCoun

<=

20

Figure 8.9: The **Breakpoint Condition** dialog box

8.2 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. Chapters 12 and 13 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 Figure 8.10. 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.

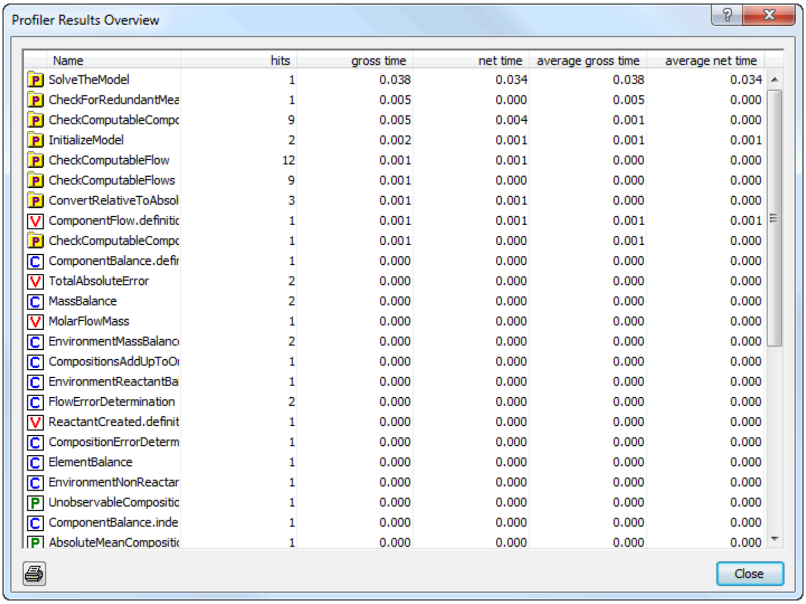
Detailed timing information

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.

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.



Profiler Results Overview

Name	hits	gross time	net time	average gross time	average net time
P SolveTheModel	1	0.038	0.034	0.038	0.034
P CheckForRedundantMea	1	0.005	0.000	0.005	0.000
P CheckComputableCompc	9	0.005	0.004	0.001	0.000
P InitializeModel	2	0.002	0.001	0.001	0.001
P CheckComputableFlow	12	0.001	0.001	0.000	0.000
P CheckComputableFlows	9	0.001	0.000	0.000	0.000
P ConvertRelativeToAbsol	3	0.001	0.001	0.000	0.000
V ComponentFlow.definitic	1	0.001	0.001	0.001	0.001
P CheckComputableCompc	1	0.001	0.000	0.001	0.000
C ComponentBalance.defir	1	0.000	0.000	0.000	0.000
V TotalAbsoluteError	2	0.000	0.000	0.000	0.000
C MassBalance	2	0.000	0.000	0.000	0.000
V MolarFlowMass	1	0.000	0.000	0.000	0.000
C EnvironmentMassBalanc	2	0.000	0.000	0.000	0.000
C CompositionsAddUpToOı	1	0.000	0.000	0.000	0.000
C EnvironmentReactantBa	1	0.000	0.000	0.000	0.000
C FlowErrorDetermination	2	0.000	0.000	0.000	0.000
V ReactantCreated.definit	1	0.000	0.000	0.000	0.000
C CompositionErrorDeterm	1	0.000	0.000	0.000	0.000
C ElementBalance	1	0.000	0.000	0.000	0.000
C EnvironmentNonReactar	1	0.000	0.000	0.000	0.000
P UnobservableCompositic	1	0.000	0.000	0.000	0.000
C ComponentBalance.inde	1	0.000	0.000	0.000	0.000
P AbsoluteMeanCompositic	1	0.000	0.000	0.000	0.000

Figure 8.10: The **Profiler Results Overview** dialog box

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.

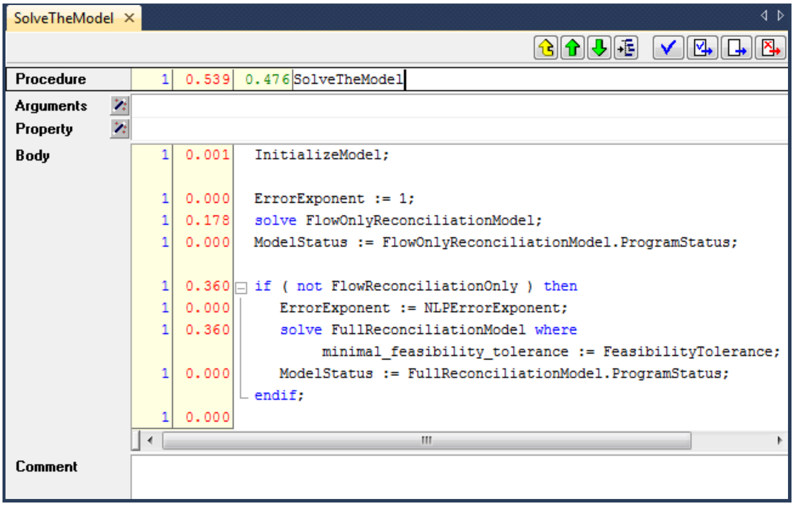
Locating offending statements

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 Figure 8.11. Similarly, AIMMS will add these profiling columns to the definition attributes of defined identifiers.

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



```
Procedure  1  0.539  0.476 SolveTheModel

Body       1  0.001   InitializeModel;

           1  0.000   ErrorExponent := 1;
           1  0.178   solve FlowOnlyReconciliationModel;
           1  0.000   ModelStatus := FlowOnlyReconciliationModel.ProgramStatus;

           1  0.360   if ( not FlowReconciliationOnly ) then
           1  0.000       ErrorExponent := NLPErrorExponent;
           1  0.360       solve FullReconciliationModel where
                              minimal_feasibility_tolerance := FeasibilityTolerance;
           1  0.000       ModelStatus := FullReconciliationModel.ProgramStatus;
                      endif;
           1  0.000
```

Figure 8.11: Profiling information in an attribute form

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.

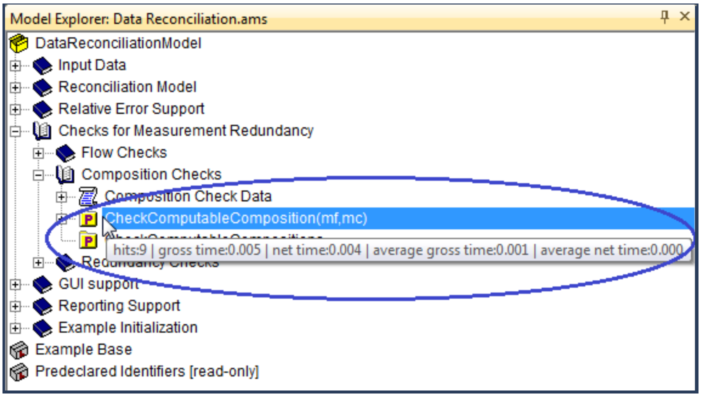
Gross versus net time for particular statements

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 Section 8.3 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 Section 15.3 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 Figure 8.12. 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 Figure 8.13).



hits:9 | gross time:0.005 | net time:0.004 | average gross time:0.001 | average net time:0.000

Figure 8.12: Observing profiling information through a tooltip

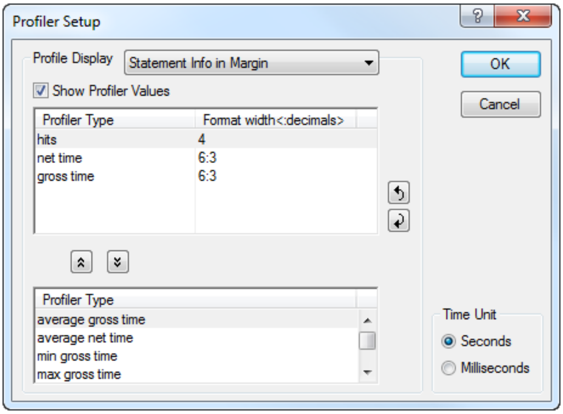
Profiler listing

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.

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 Figure 8.13. 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.



Profiler Setup

Profile Display: Statement Info in Margin

☑ Show Profiler Values

Profiler Type	Format width<:decimals>
hits	4
net time	6:3
gross time	6:3

Profiler Type
average gross time
average net time
min gross time
max gross time

Time Unit

● Seconds

○ Milliseconds

Figure 8.13: The **Profiler Setup** dialog box

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.

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.

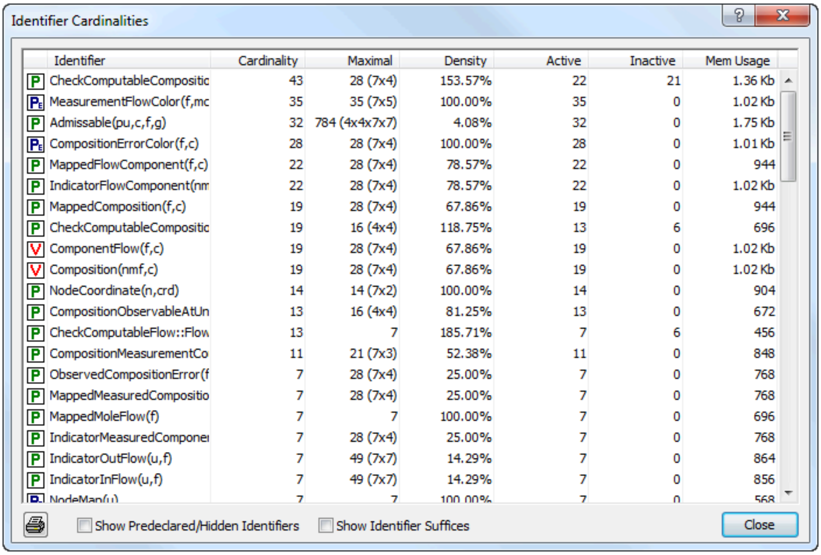
8.3 Observing identifier cardinalities

Observing identifier cardinalities

Another possible cause of performance problems is when one or more multidimensional 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 Figure 8.14. You can invoke it through the **Tools-Diagnostic Tools-Identifier Cardinalities** menu.



	Identifier	Cardinality	Maximal	Density	Active	Inactive	Mem Usage
P	CheckComputableCompositic	43	28 (7x4)	153.57%	22	21	1.36 Kb
PE	MeasurementFlowColor(f,mc	35	35 (7x5)	100.00%	35	0	1.02 Kb
P	Admissable(pu,c,f,g)	32	784 (4x4x7x7)	4.08%	32	0	1.75 Kb
PE	CompositionErrorColor(f,c)	28	28 (7x4)	100.00%	28	0	1.01 Kb
P	MappedFlowComponent(f,c)	22	28 (7x4)	78.57%	22	0	944
P	IndicatorFlowComponent(nm	22	28 (7x4)	78.57%	22	0	1.02 Kb
P	MappedComposition(f,c)	19	28 (7x4)	67.86%	19	0	944
P	CheckComputableCompositic	19	16 (4x4)	118.75%	13	6	696
V	ComponentFlow(f,c)	19	28 (7x4)	67.86%	19	0	1.02 Kb
V	Composition(nmf,c)	19	28 (7x4)	67.86%	19	0	1.02 Kb
P	NodeCoordinate(n,crd)	14	14 (7x2)	100.00%	14	0	904
P	CompositionObservableAtUn	13	16 (4x4)	81.25%	13	0	672
P	CheckComputableFlow::Flow	13	7	185.71%	7	6	456
P	CompositionMeasurementCo	11	21 (7x3)	52.38%	11	0	848
P	ObservedCompositionError(f	7	28 (7x4)	25.00%	7	0	768
P	MappedMeasuredCompositio	7	28 (7x4)	25.00%	7	0	768
P	MappedMoleFlow(f)	7	7	100.00%	7	0	696
P	IndicatorMeasuredComponei	7	28 (7x4)	25.00%	7	0	768
P	IndicatorOutFlow(u,f)	7	49 (7x7)	14.29%	7	0	864
P	IndicatorInFlow(u,f)	7	49 (7x7)	14.29%	7	0	856
P	NodeMap(u)	7	7	100.00%	7	0	568

□ Show Predeclared/Hidden Identifiers □ Show Identifier Suffices

Figure 8.14: 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 Chapter 34 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 Section 25.3 of the Language Reference).

Chapter 9

The Math Program Inspector

This chapter

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.

9.1 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 ,

9.2 Functional overview

This section

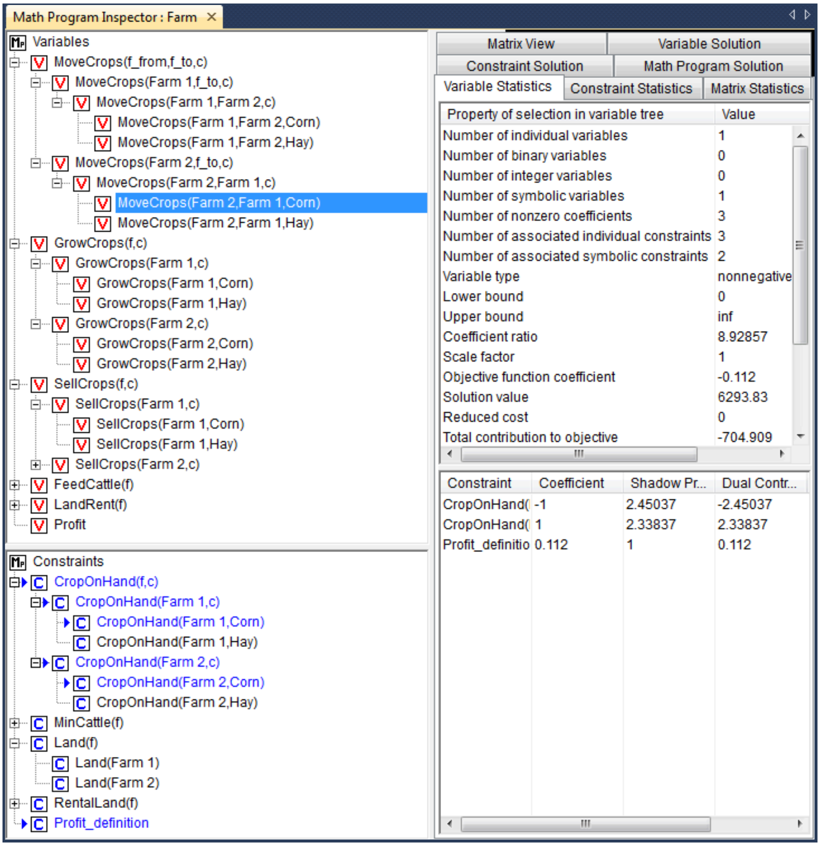
In this section you will find a description of the functionality available in the mathematical program inspector. Successively, you will learn

- 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.

9.2.1 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 Figure 9.1). 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.



Property of selection in variable tree	Value
Number of individual variables	1
Number of binary variables	0
Number of integer variables	0
Number of symbolic variables	1
Number of nonzero coefficients	3
Number of associated individual constraints	3
Number of associated symbolic constraints	2
Variable type	nonnegative
Lower bound	0
Upper bound	inf
Coefficient ratio	8.92857
Scale factor	1
Objective function coefficient	-0.112
Solution value	6293.83
Reduced cost	0
Total contribution to objective	-704.909

Constraint	Coefficient	Shadow Pr...	Dual Contr...
CropOnHand(	-1	2.45037	-2.45037
CropOnHand(	1	2.33837	2.33837
Profit_definitio	0.112	1	0.112

Figure 9.1: The math program inspector window

Tree view selections

The math program inspector contains several tabs (see the right portion of Figure 9.1) 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 `Shift` or `Control` 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 Figure 9.1 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.

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. Figure 9.1 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 Figure 9.1 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 Figure 9.1, the associated constraint selection for the selected variable has been bookmarked in the constraint tree.

9.2.2 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. Figure 9.1 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.

9.2.3 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 Figure 9.1). 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. Figure 9.2 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.

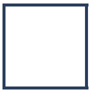


Figure 9.2: The math program inspector window

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

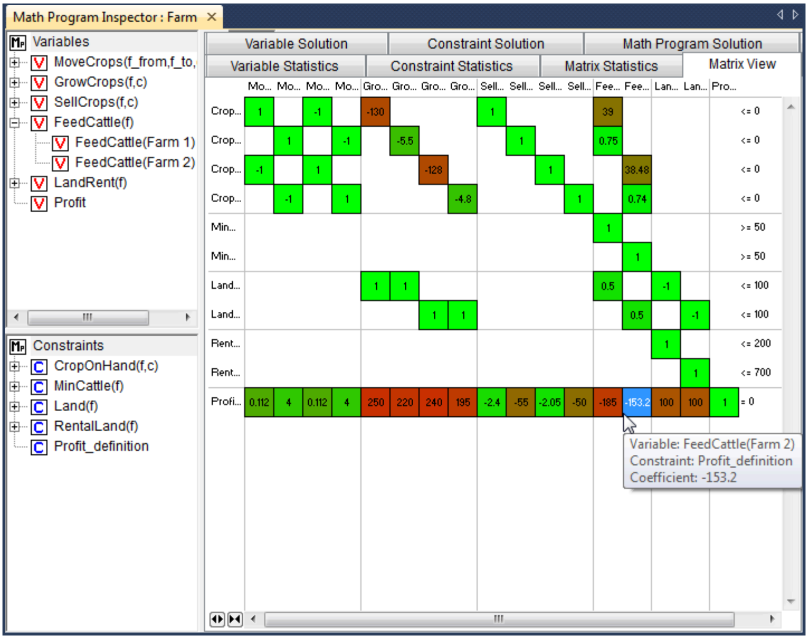
As is the case on the Variable Statistics Tab, all coefficients corresponding to nonlinear terms are denoted between square brackets. For these coefficients, the last column displays all terms that contribute to the linearized coefficient value.

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.

Matrix Statistics 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.

Matrix View tab



	Mo...	Mo...	Mo...	Mo...	Gro...	Gro...	Gro...	Gro...	Sell...	Sell...	Sell...	Sell...	Fee...	Fee...	Lan...	Lan...	Pro...	
Crop...	1		-1		-130				1				39					<= 0
Crop...		1		-1		-5.5				1			0.75					<= 0
Crop...	-1		1				-128				1			38.48				<= 0
Crop...		-1		1				-4.8				1		0.74				<= 0
Min...													1					>= 50
Min...														1				>= 50
Land...					1	1							0.5		-1			<= 100
Land...							1	1						0.5		-1		<= 100
Rent...															1			<= 200
Rent...																1		<= 700
Profi...	0.112	4	0.112	4	250	220	240	195	-2.4	-55	-2.05	-50	-185	-153.2	100	100	1	= 0

Variable: FeedCattle(Farm 2)
Constraint: Profit_definition
Coefficient: -153.2

Figure 9.3: The matrix view (individual mode)

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 $x \in (-\infty, -1000] \ \cup$

Block coloring

$[-\frac{1}{1000}, \frac{1}{1000}] \cup [1000, \infty)$ 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.

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.

9.2.4 Inspecting solution information

Solution tabs

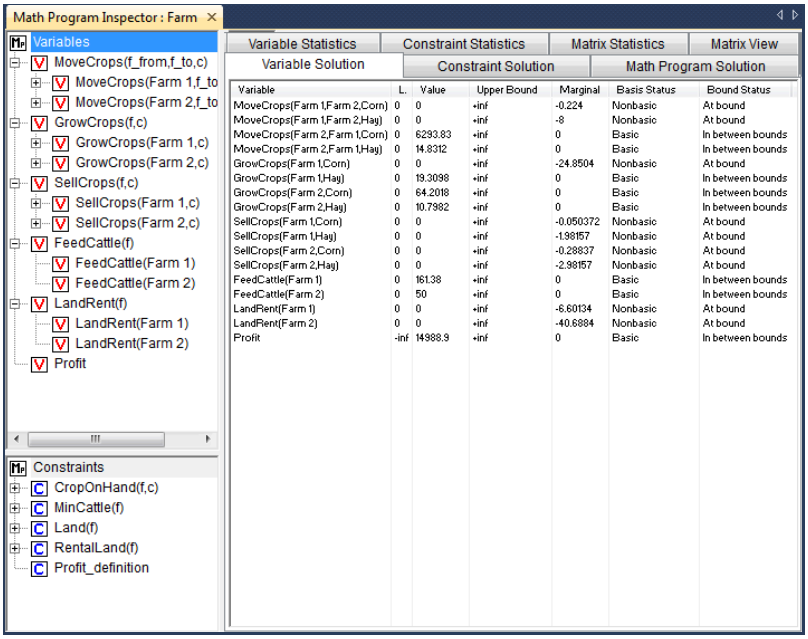
The tabs discussed so far are available as long as the math program has been generated. As soon as a solution is available, the next three tabs reveal more details about this solution.

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.



Variable	L.	Value	Upper Bound	Marginal	Basis Status	Bound Status
MoveCrops(Farm 1,Farm 2,Corn)	0	0	+inf	-0.224	Nonbasic	At bound
MoveCrops(Farm 1,Farm 2,Hay)	0	0	+inf	-8	Nonbasic	At bound
MoveCrops(Farm 2,Farm 1,Corn)	0	6293.83	+inf	0	Basic	In between bounds
MoveCrops(Farm 2,Farm 1,Hay)	0	14.8312	+inf	0	Basic	In between bounds
GrowCrops(Farm 1,Corn)	0	0	+inf	-24.8504	Nonbasic	At bound
GrowCrops(Farm 1,Hay)	0	19.3098	+inf	0	Basic	In between bounds
GrowCrops(Farm 2,Corn)	0	64.2018	+inf	0	Basic	In between bounds
GrowCrops(Farm 2,Hay)	0	10.7982	+inf	0	Basic	In between bounds
SellCrops(Farm 1,Corn)	0	0	+inf	-0.050372	Nonbasic	At bound
SellCrops(Farm 1,Hay)	0	0	+inf	-1.98157	Nonbasic	At bound
SellCrops(Farm 2,Corn)	0	0	+inf	-0.28837	Nonbasic	At bound
SellCrops(Farm 2,Hay)	0	0	+inf	-2.98157	Nonbasic	At bound
FeedCattle(Farm 1)	0	161.38	+inf	0	Basic	In between bounds
FeedCattle(Farm 2)	0	50	+inf	0	Basic	In between bounds
LandRent(Farm 1)	0	0	+inf	-6.60134	Nonbasic	At bound
LandRent(Farm 2)	0	0	+inf	-40.6884	Nonbasic	At bound
Profit	-inf	14988.9	+inf	0	Basic	In between bounds

Figure 9.4: The variable solution

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.

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

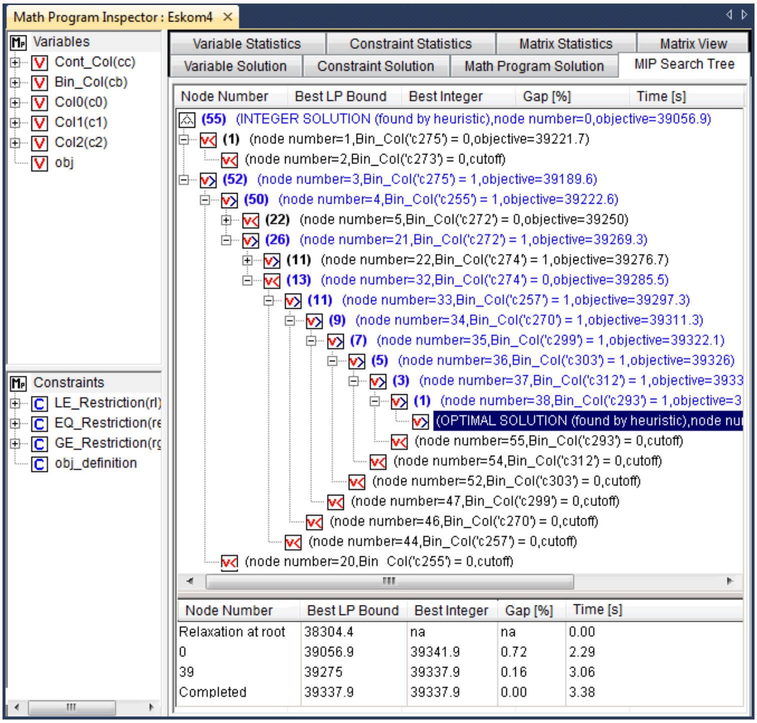
The lower part of the information retrieved by this tab is used to display logging messages resulting from the **Bound Analysis** and **Unreferenced Identifiers** commands in the **Actions** menu.

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.



```
(55) (INTEGER SOLUTION (found by heuristic),node number=0,objective=39056.9)
  (1) (node number=1,Bin_Col('c275') = 0,objective=39221.7)
    (node number=2,Bin_Col('c273') = 0,cutoff)
  (52) (node number=3,Bin_Col('c275') = 1,objective=39189.6)
    (50) (node number=4,Bin_Col('c255') = 1,objective=39222.6)
      (22) (node number=5,Bin_Col('c272') = 0,objective=39250)
      (26) (node number=21,Bin_Col('c272') = 1,objective=39269.3)
        (11) (node number=22,Bin_Col('c274') = 1,objective=39276.7)
        (13) (node number=32,Bin_Col('c274') = 0,objective=39285.5)
          (11) (node number=33,Bin_Col('c257') = 1,objective=39297.3)
            (9) (node number=34,Bin_Col('c270') = 1,objective=39311.3)
              (7) (node number=35,Bin_Col('c299') = 1,objective=39322.1)
                (5) (node number=36,Bin_Col('c303') = 1,objective=39326)
                  (3) (node number=37,Bin_Col('c312') = 1,objective=3933
                    (1) (node number=38,Bin_Col('c293') = 1,objective=3
                      (OPTIMAL SOLUTION (found by heuristic),node nu
                    (node number=55,Bin_Col('c293') = 0,cutoff)
                  (node number=54,Bin_Col('c312') = 0,cutoff)
                (node number=52,Bin_Col('c303') = 0,cutoff)
              (node number=47,Bin_Col('c299') = 0,cutoff)
            (node number=46,Bin_Col('c270') = 0,cutoff)
          (node number=44,Bin_Col('c257') = 0,cutoff)
    (node number=20,Bin Col('c255') = 0,cutoff)
```

Node Number	Best LP Bound	Best Integer	Gap [%]	Time [s]
Relaxation at root	38304.4	na	na	0.00
0	39056.9	39341.9	0.72	2.29
39	39275	39337.9	0.16	3.06
Completed	39337.9	39337.9	0.00	3.38

Figure 9.5: The MIP search tree

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.

9.2.5 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 [Go94], and is presented without derivation in the next two paragraphs.

. . . for ≤ 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.

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

. . . and ≥ 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.

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

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.

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

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 ([Ch91]). 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 ([Ch91]), 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.

9.3 A worked example

This section

The example in this section is adapted from McCarl ([Mc98]), 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.

9.3.1 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).

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

Upper bound on rental

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

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

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.

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

Cattle requirement

The cattle requirement constraint ensures that every farm keeps at least a prespecified amount of cattle.

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

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.

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

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 Figure 9.3. The solution of the particular farm model instance has already been presented in Figure 9.4.

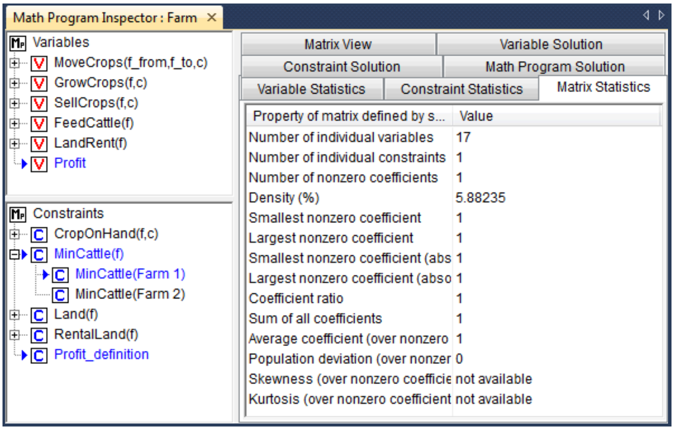
9.3.2 Investigating infeasibility

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).



Property of matrix defined by s...	Value
Number of individual variables	17
Number of individual constraints	1
Number of nonzero coefficients	1
Density (%)	5.88235
Smallest nonzero coefficient	1
Largest nonzero coefficient	1
Smallest nonzero coefficient (abs	1
Largest nonzero coefficient (abso	1
Coefficient ratio	1
Sum of all coefficients	1
Average coefficient (over nonzero	1
Population deviation (over nonzer	0
Skewness (over nonzero coefficie	not available
Kurtosis (over nonzero coefficient	not available

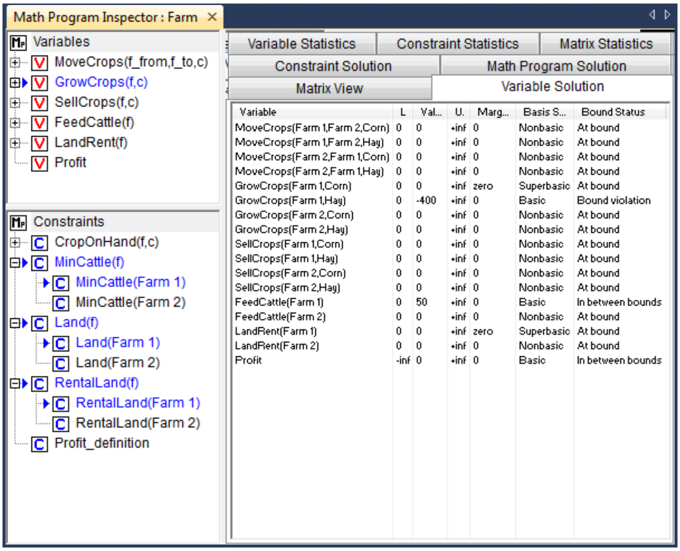
Figure 9.6: An identified substructure causing infeasibility

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. Figure 9.6 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.

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 Figure 9.7).



Variable	L	Val...	U.	Marg...	Basis S...	Bound Status
MoveCrops(Farm 1,Farm 2,Corn)	0	0	+inf	0	Nonbasic	At bound
MoveCrops(Farm 1,Farm 2,Hay)	0	0	+inf	0	Nonbasic	At bound
MoveCrops(Farm 2,Farm 1,Corn)	0	0	+inf	0	Nonbasic	At bound
MoveCrops(Farm 2,Farm 1,Hay)	0	0	+inf	0	Nonbasic	At bound
GrowCrops(Farm 1,Corn)	0	0	+inf	zero	Superbasic	At bound
GrowCrops(Farm 1,Hay)	0	-400	+inf	0	Basic	Bound violation
GrowCrops(Farm 2,Corn)	0	0	+inf	0	Nonbasic	At bound
GrowCrops(Farm 2,Hay)	0	0	+inf	0	Nonbasic	At bound
SellCrops(Farm 1,Corn)	0	0	+inf	0	Nonbasic	At bound
SellCrops(Farm 1,Hay)	0	0	+inf	0	Nonbasic	At bound
SellCrops(Farm 2,Corn)	0	0	+inf	0	Nonbasic	At bound
SellCrops(Farm 2,Hay)	0	0	+inf	0	Nonbasic	At bound
FeedCattle(Farm 1)	0	50	+inf	0	Basic	In between bounds
FeedCattle(Farm 2)	0	0	+inf	0	Nonbasic	At bound
LandRent(Farm 1)	0	0	+inf	zero	Superbasic	At bound
LandRent(Farm 2)	0	0	+inf	0	Nonbasic	At bound
Profit	-inf	0	+inf	0	Basic	In between bounds

Figure 9.7: 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.

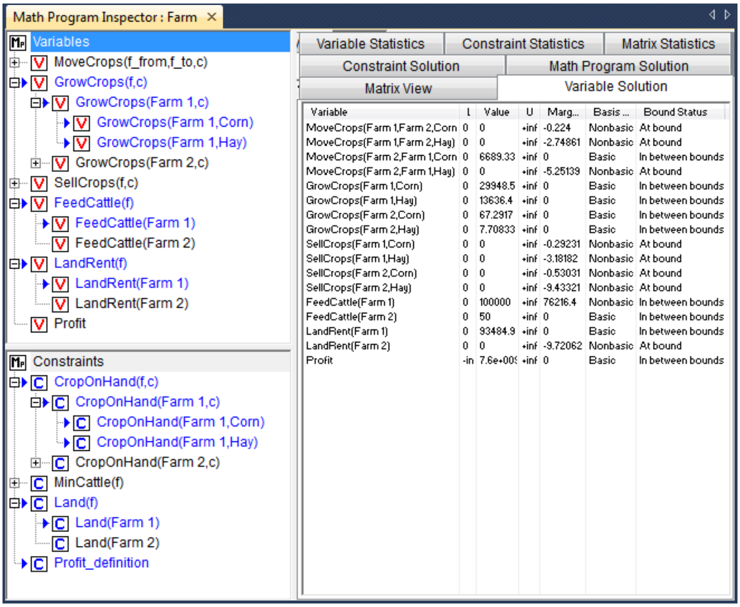
9.3.3 Investigating unboundedness

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 Figure 9.8.



Variable	L	Value	U	Marg...	Basis ...	Bound Status
MoveCrops(Farm 1,Farm 2,Corn	0	0	+inf	-0.224	Nonbasic	At bound
MoveCrops(Farm 1,Farm 2,Hay)	0	0	+inf	-2.74861	Nonbasic	At bound
MoveCrops(Farm 2,Farm 1,Corn	0	6689.33	+inf	0	Basic	In between bounds
MoveCrops(Farm 2,Farm 1,Hay)	0	0	+inf	-5.25139	Nonbasic	At bound
GrowCrops(Farm 1,Corn)	0	29948.5	+inf	0	Basic	In between bounds
GrowCrops(Farm 1,Hay)	0	13636.4	+inf	0	Basic	In between bounds
GrowCrops(Farm 2,Corn)	0	67.2917	+inf	0	Basic	In between bounds
GrowCrops(Farm 2,Hay)	0	7.70833	+inf	0	Basic	In between bounds
SellCrops(Farm 1,Corn)	0	0	+inf	-0.29231	Nonbasic	At bound
SellCrops(Farm 1,Hay)	0	0	+inf	-3.18182	Nonbasic	At bound
SellCrops(Farm 2,Corn)	0	0	+inf	-0.53031	Nonbasic	At bound
SellCrops(Farm 2,Hay)	0	0	+inf	-9.43321	Nonbasic	At bound
FeedCattle(Farm 1)	0	100000	+inf	76216.4	Nonbasic	In between bounds
FeedCattle(Farm 2)	0	50	+inf	0	Basic	In between bounds
LandRent(Farm 1)	0	93484.9	+inf	0	Basic	In between bounds
LandRent(Farm 2)	0	0	+inf	-9.72062	Nonbasic	At bound
Profit	-in	7.6e+00	+inf	0	Basic	In between bounds

Figure 9.8: An identified substructure causing unboundedness

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.

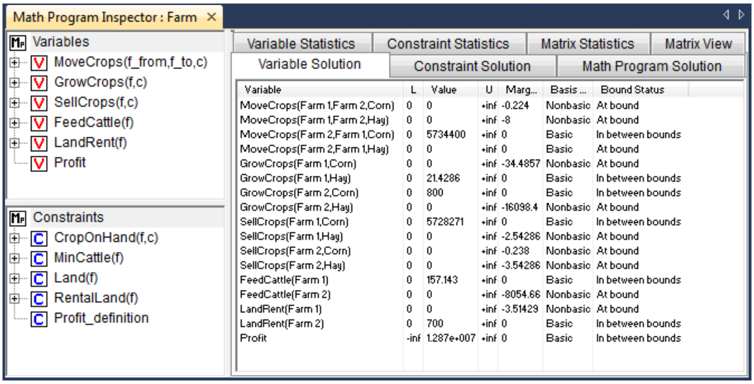
9.3.4 Analyzing an unrealistic solution

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 Figure 9.9. 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’.



Variable	L	Value	U	Marg...	Basis ...	Bound Status
MoveCrops(Farm 1,Farm 2,Corn)	0	0	+inf	-0.224	Nonbasic	At bound
MoveCrops(Farm 1,Farm 2,Hay)	0	0	+inf	-8	Nonbasic	At bound
MoveCrops(Farm 2,Farm 1,Corn)	0	5734400	+inf	0	Basic	In between bounds
MoveCrops(Farm 2,Farm 1,Hay)	0	0	+inf	0	Basic	At bound
GrowCrops(Farm 1,Corn)	0	0	+inf	-34.4857	Nonbasic	At bound
GrowCrops(Farm 1,Hay)	0	21.4286	+inf	0	Basic	In between bounds
GrowCrops(Farm 2,Corn)	0	800	+inf	0	Basic	In between bounds
GrowCrops(Farm 2,Hay)	0	0	+inf	-16098.4	Nonbasic	At bound
SellCrops(Farm 1,Corn)	0	5728271	+inf	0	Basic	In between bounds
SellCrops(Farm 1,Hay)	0	0	+inf	-2.54286	Nonbasic	At bound
SellCrops(Farm 2,Corn)	0	0	+inf	-0.238	Nonbasic	At bound
SellCrops(Farm 2,Hay)	0	0	+inf	-3.54286	Nonbasic	At bound
FeedCattle(Farm 1)	0	157.143	+inf	0	Basic	In between bounds
FeedCattle(Farm 2)	0	0	+inf	-8054.66	Nonbasic	At bound
LandRent(Farm 1)	0	0	+inf	-3.51429	Nonbasic	At bound
LandRent(Farm 2)	0	700	+inf	0	Basic	In between bounds
Profit	-inf	1.287e+007	+inf	0	Basic	In between bounds

Figure 9.9: An unrealistic solution

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 Figure 9.10). 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.

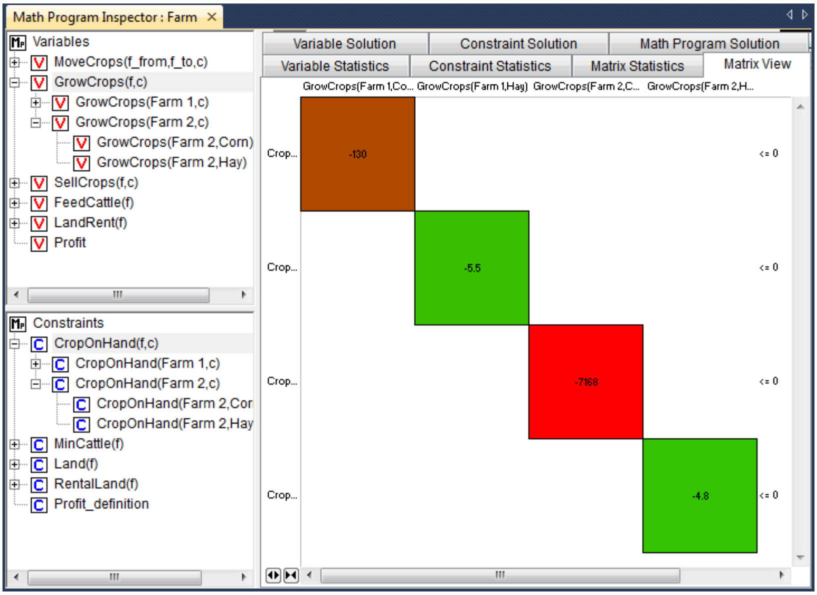


Figure 9.10: The Matrix View tab for an unrealistic solution

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

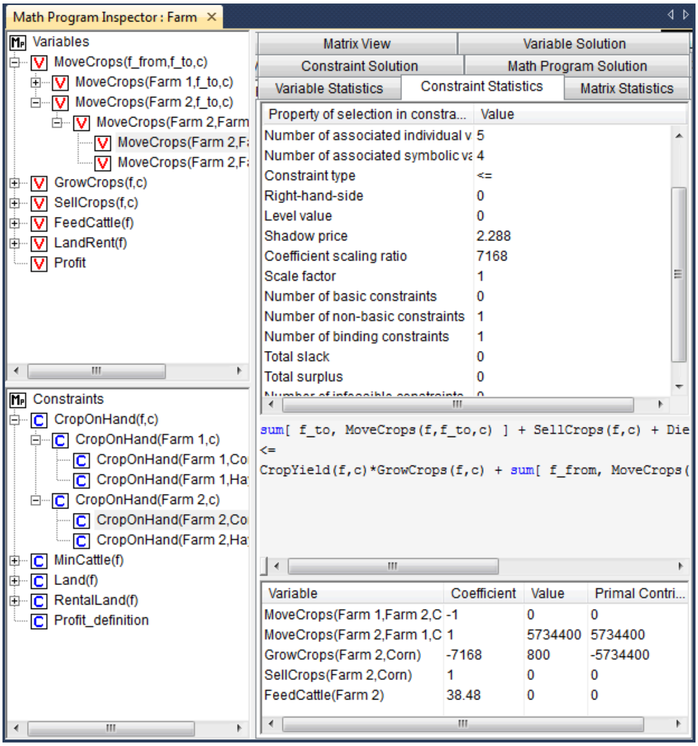
The following iterative procedure can be followed to resolve an unrealistic solution.

- 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



Property of selection in constra...	Value
Number of associated individual v	5
Number of associated symbolic va	4
Constraint type	<=
Right-hand-side	0
Level value	0
Shadow price	2.288
Coefficient scaling ratio	7168
Scale factor	1
Number of basic constraints	0
Number of non-basic constraints	1
Number of binding constraints	1
Total slack	0
Total surplus	0

```
sum[ f_to, MoveCrops(f,f_to,c) ] + SellCrops(f,c) + Die
<=
CropYield(f,c)*GrowCrops(f,c) + sum[ f_from, MoveCrops(
```

Variable	Coefficient	Value	Primal Contri...
MoveCrops(Farm 1,Farm 2,C	-1	0	0
MoveCrops(Farm 2,Farm 1,C	1	5734400	5734400
GrowCrops(Farm 2,Corn)	-7168	800	-5734400
SellCrops(Farm 2,Corn)	1	0	0
FeedCattle(Farm 2)	38.48	0	0

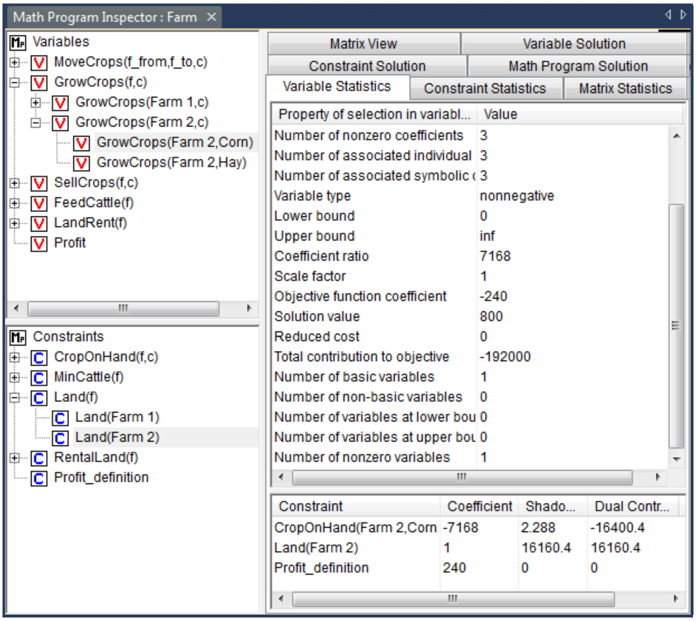
Figure 9.11: Inspecting primal contributions

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 Figure 9.11).

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 Figure 9.12).



Property of selection in variabl...	Value
Number of nonzero coefficients	3
Number of associated individual	3
Number of associated symbolic (	3
Variable type	nonnegative
Lower bound	0
Upper bound	inf
Coefficient ratio	7168
Scale factor	1
Objective function coefficient	-240
Solution value	800
Reduced cost	0
Total contribution to objective	-192000
Number of basic variables	1
Number of non-basic variables	0
Number of variables at lower bou	0
Number of variables at upper bou	0
Number of nonzero variables	1

Constraint	Coefficient	Shado...	Dual Contr...
CropOnHand(Farm 2,Corn	-7168	2.288	-16400.4
Land(Farm 2)	1	16160.4	16160.4
Profit_definition	240	0	0

Figure 9.12: Inspecting dual contributions

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.

Bibliography

[]Ch91 J.W. Chinneck, *Locating minimal infeasible constraint sets in linear programs*, ORSA Journal on Computing, vol. 3, (1991), no. 1, 157–168.

[]Go94 J. Gondzio, *Presolve analysis of linear programs prior to applying an interior point method*, Logilab, HEC Geneva, Section of Management Studies, University of Geneva, Technical Report 1994.3 (1994)

[]Mc98 B.A. McCarl, *A note on fixing misbehaving mathematical programs: Post-optimality procedures and GAMS-related software*, Journal of Agricultural & Applied Economics, vol. 30, (1998), no. 2, 403–414.

Part III

Creating an End-User Interface

Chapter 10

Pages and Page Objects

Creating an end-user interface

After you have created a model in AIMMS to represent and solve a particular problem, you may want to move on to the next step: creating a graphical end-user interface around the model. In this way, you and your end-users are freed from having to enter (or alter) the model data in text or database tables. Instead, they can make the necessary modifications in a graphical environment that best suits the purposes of your model. Similarly, using the advanced graphical objects available in AIMMS (such as the Gantt chart and network flow object), you can present your model results in an intuitive manner, which will help your end-users interpret a solution quickly and easily.

This chapter

This chapter gives you an overview of the possibilities that AIMMS offers you for creating a complete model-based end-user application. It describes pages, which are the basic medium in AIMMS for displaying model input and output in a graphical manner. In addition, the chapter illustrates how page objects (which provide a graphical display of one or more identifiers in your model) can be created and linked together.

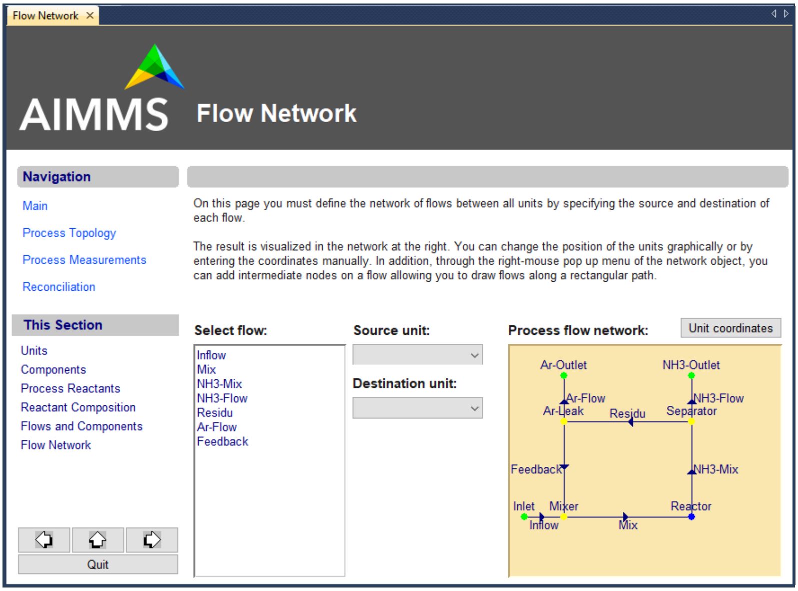
10.1 Introduction

What is a page?

A *page* is a window in which the data of an AIMMS model is presented in a graphical manner. Pages are the main component of an end-user interface for a model-based decision support application. An example of an end-user page is given in Figure 10.1. The page shown here provides a comprehensive graphical overview of the results of an optimization model by means of a *network flow object* in which flows which require attention are colored red. By clicking on a particular flow in the network object, additional information about that flow is shown in the tables on the left of the page.

Page design

Pages are fully designed by application developers for use by the end-users of an application. Thus, you, as a developer, can decide at what position in the interface particular model data should be presented to the end-user, and in which format. In addition, by automatically executing procedures when opening or closing a page or when modifying data, you can make sure that



Flow Network

AIMMS Flow Network

Navigation

Main

Process Topology

Process Measurements

Reconciliation

This Section

Units

Components

Process Reactants

Reactant Composition

Flows and Components

Flow Network

On this page you must define the network of flows between all units by specifying the source and destination of each flow.

The result is visualized in the network at the right. You can change the position of the units graphically or by entering the coordinates manually. In addition, through the right-mouse pop up menu of the network object, you can add intermediate nodes on a flow allowing you to draw flows along a rectangular path.

Select flow:

Inflow
Mix
NH3-Mix
NH3-Flow
Residu
Ar-Flow
Feedback

Source unit:

Destination unit:

Process flow network:

Figure 10.1: Example of a end-user page

all the necessary computations are performed before certain model results are displayed.

10.2 Creating pages

Creating pages

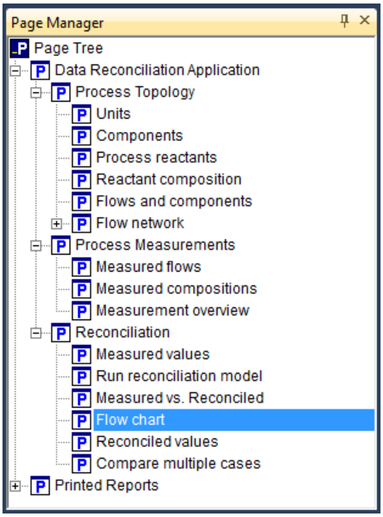
Creating an end-user page is as easy as adding a new node to the page tree in the **Page Manager** (see Chapter 12). Figure 10.2 illustrates the page tree associated with the example application used throughout this guide. As all the trees in the AIMMS modeling tools work alike, you can use any of the methods described in Section 4.3 to add a new page node to the page tree.

Copying pages

In addition to inserting a new empty page into the page tree, you can also copy an existing page or an entire subtree of pages, by either a simple cut, copy and paste or a drag-and-drop action in the tree (see Section 4.3). All copied pages will have the same content as their originals.

Page name

The node name of every page (as displayed in the page tree) is the unique name or description by which the page is identified in the system. When you add new pages to the tree, AIMMS will name these *Page 1*, *Page 2*, etc. You can change this name using the standard methods for changing names of tree nodes as described in Section 4.3.



Page Manager

- Page Tree
- Data Reconciliation Application
  - Process Topology
    - Units
    - Components
    - Process reactants
    - Reactant composition
    - Flows and components
    - Flow network
  - Process Measurements
    - Measured flows
    - Measured compositions
    - Measurement overview
  - Reconciliation
    - Measured values
    - Run reconciliation model
    - Measured vs. Reconciled
    - Flow chart
    - Reconciled values
    - Compare multiple cases
- Printed Reports

Figure 10.2: Example of a page tree

Page title

By default, the node name is the title that will be displayed in the frame of the page window when the page is opened. In the page **Properties** dialog box (see Section 11.2) you can, however, specify a different page title to be displayed, which can either be a constant string or a reference to a string parameter in the model. The latter is useful, for instance, if you intend to set up an end-user interface in multiple languages.

Position in page tree

Its position in the page tree determines the navigational properties of the page. It will determine how any button with references to the next or previous page, or any navigation object or menu linked to the page, will behave. These navigational aspects of the **Page Manager** tool are discussed in more detail in Chapter 12.

Using templates

Every page that you add to the page tree, is also automatically added to the template tree in the **Template Manager**. By moving the page to a different position in the template tree, the page automatically inherits all the properties such as page size or background, and all objects specified on the template pages hierarchically above it. The **Template Manager** and the use of templates is explained in full detail in Chapter 12.

10.3 Adding page objects

Page objects

All visible components on a page are instances of the collection of *page objects* as offered by AIMMS. Such page objects are mostly used to visualize the input and output data of your model in various ways. They also include simple drawing objects, such as lines and circles, and buttons for navigation and execution.

Edit mode

Before you can add page objects to a page, the page must be in *edit mode*. When you open a page using the **Page Manager**, it is opened in *user mode* by default. When you want to open a page in edit mode from the **Page Manager**, you can do so using the right mouse pop-up menu. If a page is already opened in user mode, you can reopen it in edit mode using the button on the page toolbar. When you open the page from the **Template Manager**, it is opened in edit mode by default.

Common data objects

AIMMS provides the most common graphical data objects such as

- row-oriented composite tables,
- 2-dimensional tables,
- pivot tables,
- graphs, and
- charts.

These objects can be used both for displaying and for modifying the data in your model. The data displayed in such objects are always directly linked to one or more identifiers in your model.

Adding an object

Placing a data object onto a page can be done without any programming. The following straightforward actions are required:

- select the type of the graphical object to be displayed,
- drag a rectangle onto the page of the intended size of the object, and
- choose the identifier in the model holding the data that you want to display.

Selecting the object type

You can select the object type that you want to add to the page from the **Object** menu. Alternatively, you can select any of the most common object types using the **Page Edit** toolbar, as depicted in Figure 10.3. If you move the cursor to one of the buttons of the toolbar, a tooltip will appear. After you have selected an object type, the page cursor will change to a cross allowing you to drag the rectangle in which the object will be contained. Figure 10.4 illustrates such a rectangle just prior to linking it to one or more AIMMS identifiers.

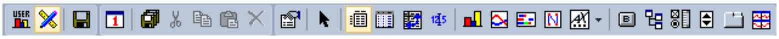


Figure 10.3: The **Page Edit** toolbar

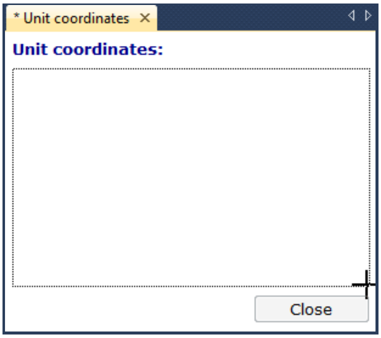


Figure 10.4: Drawing an object region

Object grid and alignment

In order to let you drag object regions in an aligned manner, AIMMS allows you to associate a grid with a page, and align object regions to that grid automatically via the **View** menu. Alternatively, you may align objects later on, or make them the same size via the **Edit-Alignment** menu (see Section 11.1).

Selecting an identifier . . .

After you have indicated the object region, you must select an identifier to be associated with that object. To support you in this task AIMMS provides an **Identifier Selection** dialog box as illustrated in Figure 10.5. You can select any single identifier from the list on the right.

. . . from a subselection

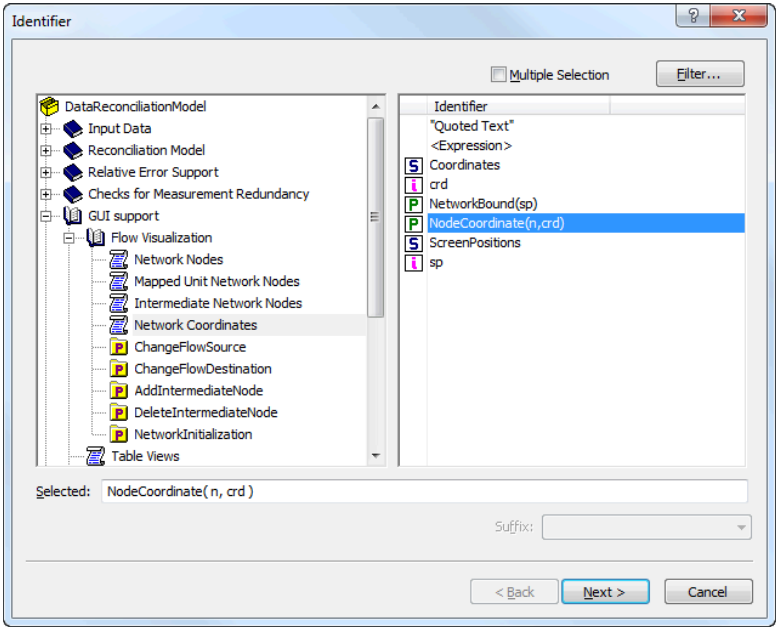
Additional help is offered for models with many identifiers. By selecting a subtree of the model tree on the left-hand side of the dialog box, you can narrow down the selection of identifiers on the right-hand side to those which are declared within the selected subtree. With the **Filter...** button you can narrow the selection down even more, by only displaying those identifier types that you are interested in.

Selecting from a library

When your project contains one or more library projects, AIMMS only allows you to select identifiers that are part of the interface of a library on any page not included in such a library (see also Section 3.2). If the page is part of the page tree of a library, AIMMS allows you to select from *all* the identifiers declared in the library.

Ensuring your freedom

By restricting access from within pages outside of the library to the identifiers in the library interface only, AIMMS allows you to freely modify the internal implementation of your library. No other part of the application will be inflicted if you make changes to identifier declarations that are not included in the library interface.



Selected: NodeCoordinate(n, crd)

Figure 10.5: The **Identifier Selection** dialog box

Slices and linking

In its simplest form, you can use the **Identifier Selection** dialog box to select an entire identifier of the appropriate dimension to fill a selected object. However, the **Identifier Selection** dialog box will also let you consider selecting *slices* of identifiers, or provide automatic *links* between objects. These advanced subjects will be discussed in detail in Section 10.4 below.

Object properties

After you have selected the identifier(s) necessary to fill the page object with the appropriate model data, AIMMS will draw the object using default settings for properties such as fonts, colors and borders. Later on, you can change these properties (or even modify the defaults) via the **Properties** dialog box of the object (see also Section 11.2).

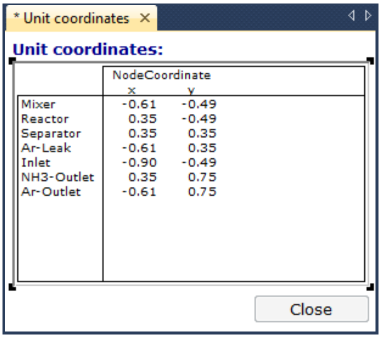
Example

If the object region displayed in Figure 10.4 is used to draw a table object, and the identifier selection dialog box in Figure 10.5 is used to select the identifier `NodeCoordinate(n,crd)`, the table in Figure 10.6 results.

10.3.1 Displaying expressions in page objects

Displaying expressions

In addition to indexed identifiers, AIMMS also allows you to display expressions in a page object. This is convenient, for instance, when you want to display

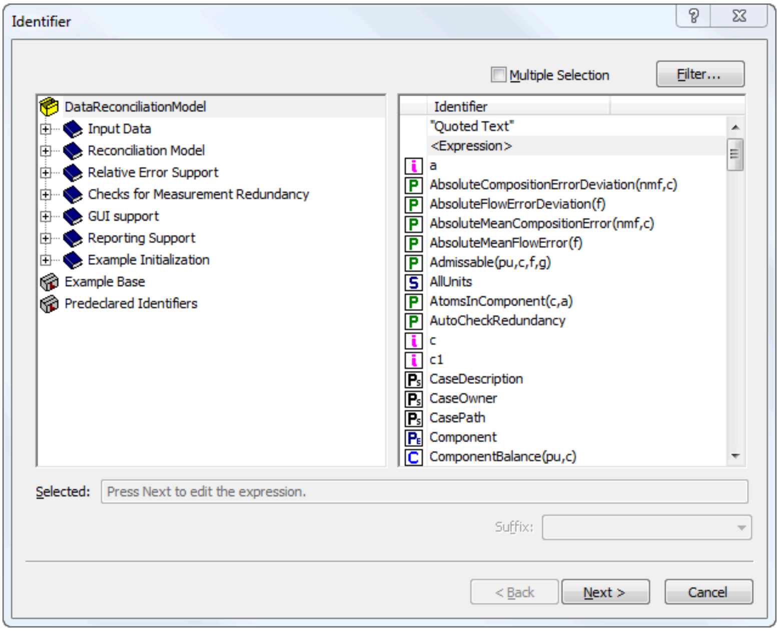


Unit coordinates:

	NodeCoordinate	
	x	y
Mixer	-0.61	-0.49
Reactor	0.35	-0.49
Separator	0.35	0.35
Ar-Leak	-0.61	0.35
Inlet	-0.90	-0.49
NH3-Outlet	0.35	0.75
Ar-Outlet	-0.61	0.75

Figure 10.6: Example of a newly created table object

some data which is not directly available in your model in the form of an (indexed) identifier, but which can be easily computed through an expression referring to one or more identifiers in your model. In such a case, you do not have to create an additional defined parameter containing the expression that you want to display, but rather you can directly enter the expression in the **Identifier Selection** dialog box, as illustrated in Figure 10.7.



- DataReconciliationModel
  - Input Data
  - Reconciliation Model
  - Relative Error Support
  - Checks for Measurement Redundancy
  - GUI support
  - Reporting Support
  - Example Initialization
- Example Base
- Predeclared Identifiers

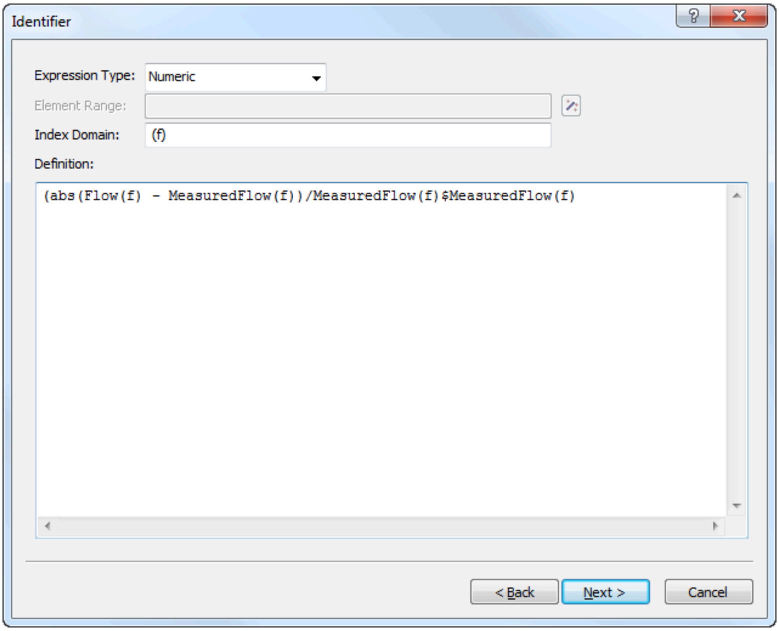
Identifier
"Quoted Text"
<Expression>
i a
P AbsoluteCompositionErrorDeviation(nmf,c)
P AbsoluteFlowErrorDeviation(f)
P AbsoluteMeanCompositionError(nmf,c)
P AbsoluteMeanFlowError(f)
P Admissable(pu,c,f,g)
S AllUnits
P AtomsInComponent(c,a)
P AutoCheckRedundancy
i c
i c1
P CaseDescription
P CaseOwner
P CasePath
P Component
C ComponentBalance(pu,c)

Selected: Press Next to edit the expression.

Figure 10.7: Selecting an expression in a page object

Entering an expression

When you have indicated that the page object should display an expression rather than an indexed identifier, AIMMS will display the **Expression Definition** dialog box illustrated in Figure 10.8. In this dialog box you must specify the exact definition of the expression you want to be displayed in the page object.



Expression Type: Numeric
Element Range:
Index Domain: (f)
Definition:

```
(abs(Flow(f) - MeasuredFlow(f))/MeasuredFlow(f)$MeasuredFlow(f)
```

Figure 10.8: Entering an expression for a page object

Specifying the expression type

In the **Expression Type** field of the **Expression Definition** dialog box, you must select the type of the expression you entered. AIMMS only allows the display of

- numeric,
- element-valued, and
- string-valued.

expressions. AIMMS does not allow the display of set expressions. If the expression type is element-valued, you must also indicate the element range of the expression, i.e. the set in which the expression will hold its values.

Specifying the index domain

Finally, in the **Index Domain** field of the **Expression Definition** dialog box you must specify the index domain over which the expression is defined. Contrary to the `IndexDomain` attribute in a parameter declaration form, AIMMS only accepts a list of indices in this field, i.e. you cannot add a domain condition (see also Section 4.1 of the Language Reference). If you want to restrict the domain of the expression, you specify the domain condition as a $ condition within the expression definition (see also Section 6.1.9 of the Language Reference). This is illustrated in Figure 10.8, where `MeasuredFlow(f)` serves as a domain condition on the domain `f`.

10.3.2 Creating advanced page objects

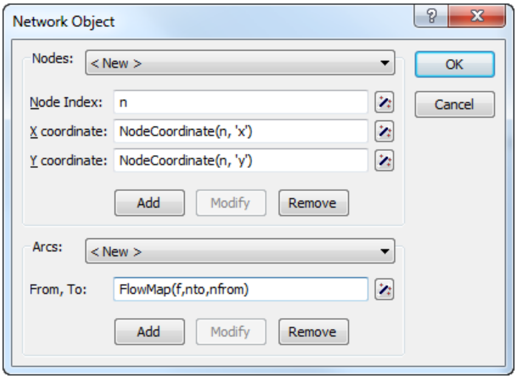
Advanced data objects . . .

In addition to common graphical data objects such as tables, bar charts and curves, AIMMS also supports a number of advanced graphical objects. These objects are designed for specialized, but widely-used, application areas. The most notable advanced objects available in AIMMS are:

- an interactive *Gantt chart* for time-phased scheduling and planning applications, and
- a *network flow object* for applications in which two-dimensional maps or flows play a central role.

. . . are based on multiple identifiers

Advanced data objects have the characteristic that multiple model identifiers are required to represent the visual result. For instance, in the network flow object you need a set identifier to denote the set of nodes to be displayed and their coordinates in the network, as well as a parameter to indicate the flow values between these nodes. Figure 10.9 illustrates the selection dialog box of a network flow object. To enter the appropriate identifiers for each required component, you can open the common **Identifier Selection** dialog box described above by pressing the wizard button at the right of each individual component.



Network Object

Nodes: < New >

Node Index: n

X coordinate: NodeCoordinate(n, 'x')

Y coordinate: NodeCoordinate(n, 'y')

Arcs: < New >

From, To: FlowMap(f,nto,nfrom)

Figure 10.9: Identifier selection for the network flow object

Object help

In this User's Guide you will only find the basic mechanisms for adding or modifying pages and page objects. Full details of all object types, and their properties and settings, are described in the on-line help facility which is always available when you are running AIMMS.

Non-data objects

In addition to data-related objects, AIMMS also supports various other types of objects such as:

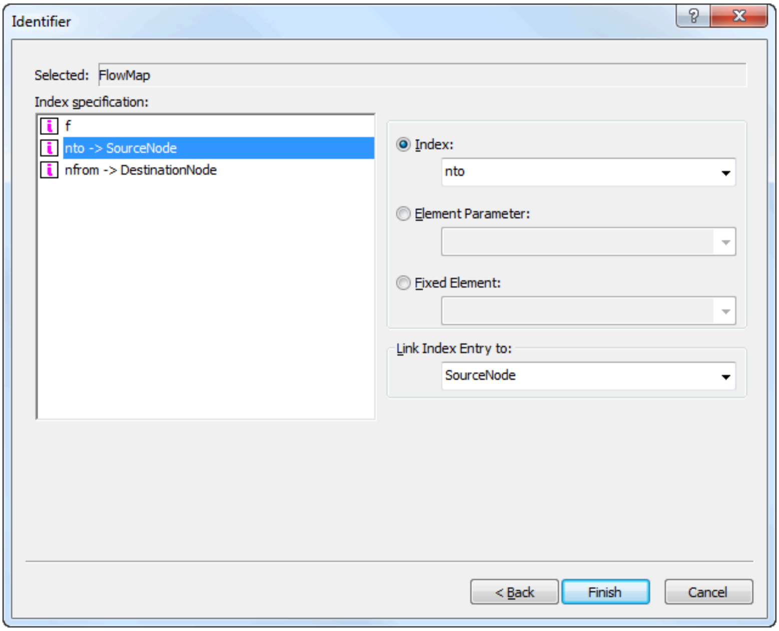
- drawing objects (such as line, circle, rectangle, picture and text objects), and
- buttons to initiate model execution and page navigation.

Drawing objects and buttons are positioned on a page in exactly the same manner as the data objects described above, except that a link to one or more AIMMS identifiers is not required.

10.4 Selecting identifier slices and linking objects

Advanced identifier selection

After you have selected an indexed identifier (or expression) in the **Identifier Selection** dialog box, a second dialog box appears, as illustrated in Figure 10.10. In this dialog box, you have several options to refine your choice,



Identifier

Selected: FlowMap

Index specification:

- f
- nto -> SourceNode
- nfrom -> DestinationNode

● Index: nto

○ Element Parameter:

○ Fixed Element:

Link Index Entry to: SourceNode

Figure 10.10: Advanced **Identifier Selection** options

each of which will be described in this section.

Slicing and subset restriction

By default, AIMMS assumes that you want to associate the full identifier with the object in hand. However, with the dialog box of Figure 10.10 AIMMS allows you to modify several domain-related issues before displaying the identifier. More specifically, for every individual dimension in the index domain of the identifier, you can:

- restrict that dimension to those elements that are included in a particular subset associated with the domain set by using a subset index,
- select a slice of the identifier by fixing that dimension to the value of a particular scalar element-valued parameter that assumes its values into the corresponding domain set, or
- select a slice of the identifier by fixing that dimension to a specific element in the corresponding domain set.

In the dialog box of Figure 10.10 AIMMS lets you select specific elements, element parameters or subset indices on the right-hand side of the dialog box to restrict the dimension that is selected on the left-hand side.

Dimension reduction

By fixing a particular dimension to an element parameter or a set element, the total number of dimensions of the displayed data is reduced by one. Thus, by fixing one dimension of a two-dimensional parameter, only a one-dimensional table will be displayed. The number of dimensions is not reduced when the display is restricted to elements in a subset. In this case, however, the object will display less data.

Index order and table split

For a table object, the **Identifier Selection** dialog box also lets you determine the order of the dimensions and a split of the dimensions. This allows you to specify which dimensions are shown rowwise and which columnwise, and in which order. If you do not insert a split manually, AIMMS will determine a default split strategy.

Index linking

Finally, the identifier selection options offer you the possibility of establishing a link between a particular dimension of the selected identifier and a (scalar) element parameter that assumes its values into the corresponding domain set. As an example, consider the dialog box of Figure 10.10. In it, the dimension `nfrom` of the identifier `FlowMap(f,nfrom,nto)` is linked to the element parameter `SourceNode`, and the dimension `nto` to the element parameter `DestinationNode`.

Link notation

In the **Properties** dialog boxes of a linked object, AIMMS displays the link using a "->" arrow. Thus, the parameter `FlowMap` from the example above, will be displayed as

```
FlowMap( f, nfrom -> SourceNode, nto -> DestinationNode )
```

This special link notation is only valid in the graphical interface, and cannot be used anywhere else in the formulation of your model.

Effect of index linking

When the identifier FlowMap(f,nfrom,nto) is displayed in, for instance, a table object, AIMMS will, as a result of the specified index links, automatically assign the values of nfrom and nto associated with the currently selected table entry to the element parameters SourceNode and DestinationNode, respectively.

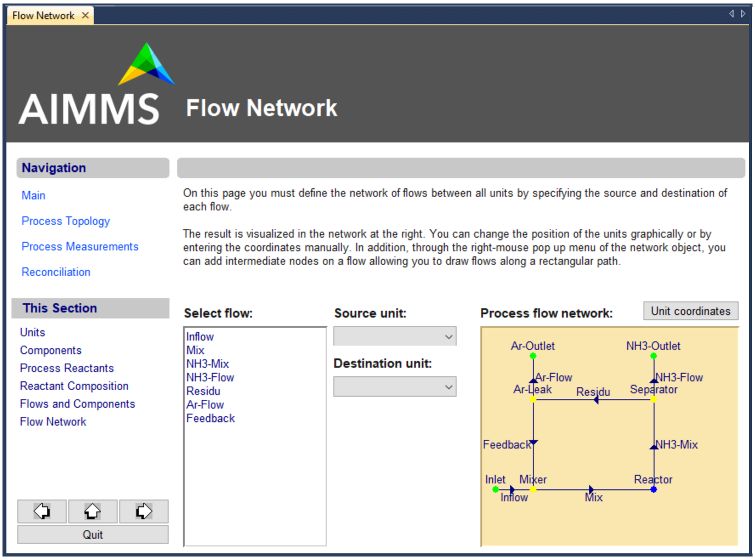
Use of index linking

Index linking is a very powerful AIMMS feature that allows you to effectively implement several attractive features in an end-user interface without any programming effort on your part. Some representative uses of index linking are discussed below.

- You can use index links involving one or more element parameters in a particular page object as a way of triggering AIMMS to automatically update one or more other page objects that contain identifier slices fixed to these element parameters. These updates will occur as soon as a user clicks somewhere in the particular page object in which the indices were linked. An illustrative example of such automatic linkage of page objects is shown below.
- You can use index linking to keep track of the current user selection in an object when executing a procedure within your model. This allows you to do some additional data processing, or perform some necessary error checks for just that tuple in a multidimensional identifier, whose value has most recently been modified by the end-user of your application.

Example

Consider the page shown in Figure 10.11. The tables and lists underneath the



On this page you must define the network of flows between all units by specifying the source and destination of each flow.

The result is visualized in the network at the right. You can change the position of the units graphically or by entering the coordinates manually. In addition, through the right-mouse pop up menu of the network object, you can add intermediate nodes on a flow allowing you to draw flows along a rectangular path.

Figure 10.11: Example of index linking

flow data text in the center part of the page display detailed information regarding the currently selected flow in the network flow object shown in the right part of the page. This was accomplished as follows. The index f representing the flows in the network flow object on the right was linked to a single element parameter FlowEl in the set Flows. The tables and lists on the left of the screen contain identifier slices fixed to the element parameter FlowEl. Take, for instance, the values in the column named Measured in the table object on the lower left part of the screen. This column corresponds to the one-dimensional identifier slice MappedMeasuredComposition(c,FlowEl). As a result of the link, the column Measured automatically displays detailed information for the flow selected by the end- user in the flow chart on the right.

Chapter 11

Page and Page Object Properties

This chapter

After you have created a page with one or more data objects on it, AIMMS allows you to modify the display properties of these objects. This chapter illustrates the available tools for placing and ordering page objects, and how to modify properties of both pages and page objects. It also provides a brief description of the available properties.

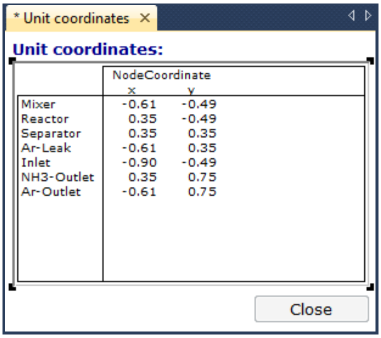
11.1 Selecting and rearranging page objects

Selecting an object

Before you can modify the properties of a page object, you must select the object. This can be accomplished as follows:

- make sure that the page is opened in edit mode (see Section 10.3),
- press the **Select Object** button ↖ on the page toolbar, if it is not already pressed, and
- click on the page object.

The selected object(s) on a page are marked with a small dark square on each of its corners. This is illustrated in Figure 11.1.



	NodeCoordinate	
	x	y
Mixer	-0.61	-0.49
Reactor	0.35	-0.49
Separator	0.35	0.35
Ar-Leak	-0.61	0.35
Inlet	-0.90	-0.49
NH3-Outlet	0.35	0.75
Ar-Outlet	-0.61	0.75

Figure 11.1: A selected page object

No template objects

When a page depends on one or more templates (see also Section 12.2), AIMMS will only let you select those objects that were placed on the page itself, and not those which are contained in any of its templates. Template objects can only be edited in the template page on which they are defined.

Selecting overlapping objects

When two or more objects are overlapping, clicking on the overlapping region will result in any one of the overlapping objects being selected. By holding the **Shift** key down during clicking, AIMMS will cycle through all the overlapping objects, allowing you to select the object of your choice. Alternatively, you can press the **Tab** key repeatedly to browse through all selectable objects on the page.

Selecting multiple objects

In addition to selecting a single page object, AIMMS also allows you to select multiple objects. You can do this by dragging a *select region* on the page, after which AIMMS will mark all objects contained in that region as selected. Alternatively, you can add or remove objects to form a selection by clicking on the objects while holding down the **Shift** key.

Object alignment

With the **Edit-Alignment** menu of a page in edit mode, you can correct the placement and sizes of all page objects that are currently selected. The **Alignment** menu lets you perform actions such as:

- give all selected objects the same height or width, i.e. the height or width of the largest object,
- align all selected objects with the top, bottom, left or rightmost selected object,
- center the selected objects horizontally or vertically, and
- spread all selected objects equally between the top and bottommost objects or between the left and rightmost objects.

An alternative method of alignment is to define a grid on the page (see Section 10.3), and align the borders of all objects with the grid.

Drawing order

With the **Drawing Order** item of the **Edit** menu, you can alter the order in which overlapping objects are drawn. When applied to a selected object, you can specify that the object at hand must be drawn as either the top or bottommost object. Modifying the drawing order only makes sense for drawing objects such as the text, rectangle, line, circle and picture objects.

Specifying the tab order

When there is a natural order in which an end-user has to enter data on a particular page, you can use the **Tab Order** item from the **Edit** menu, to specify this order. The **Tab Order** menu opens a dialog box as illustrated in Figure 11.2. In this dialog box all page objects are displayed in a list which determines the (cyclic) order in which AIMMS will select the next object for editing when the user leaves another object on the page through the **Tab** or **Enter** keys.

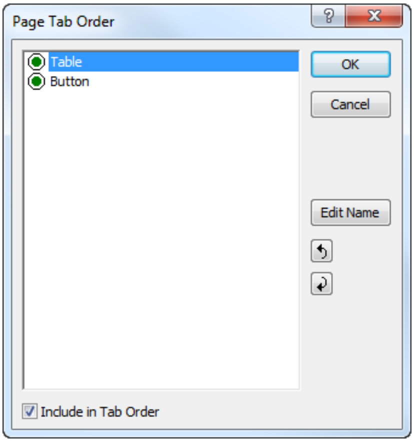


Figure 11.2: The **Tab Order** dialog box

Tabular objects

In tabular objects, the **Tab** and **Enter** keys can also be used to move to the next table entry to the right or below, respectively. In such cases, AIMMS will only go to the next object in the tab order, if further movement to the right or below within the object is no longer possible.

Disabling tab order

In addition to modifying the tab order, you can also use dialog box of Figure 11.2 to select the page objects that should not be included in the tab order. Alternatively, you can remove a page object from the tab order in the **Properties** dialog box of that object as explained in the next section. Objects excluded from the tab order are not accessible on the page by pressing the **Tab** or **Enter** keys, but can still be selected using the mouse.

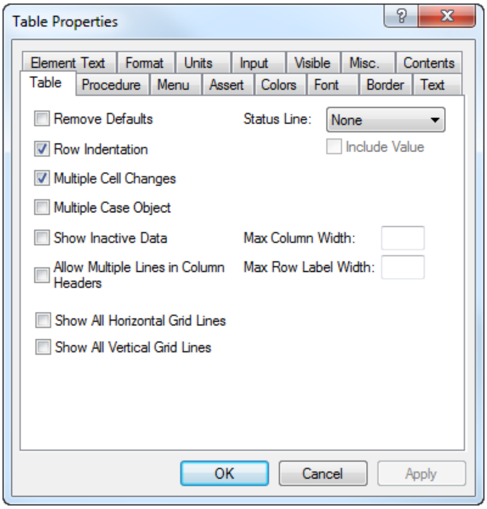
11.2 Modifying page and object properties

Object properties

In addition to modifying the display properties of groups of objects on a page, AIMMS also allows you to modify the visual appearance of a page itself and of all of its individual page objects. When the page is in edit mode, you can open the **Properties** dialog box of either a page or a page object by simply double clicking on it, or by selecting **Properties** from the right-mouse pop-up menu. This will display a dialog box as illustrated in Figure 11.3. The dialog box contains tabs for all visual aspects that are relevant to that object, and initially displays the current settings of these visual aspects.

Properties of multiple objects

You can also modify properties of multiple objects at the same time by first selecting a group of objects and then selecting the **Edit-Properties** menu, or selecting **Properties** from the right-mouse pop-up menu. This will invoke a



□ Remove Defaults
☑ Row Indentation
☑ Multiple Cell Changes
□ Multiple Case Object
□ Show Inactive Data
□ Allow Multiple Lines in Column Headers
□ Show All Horizontal Grid Lines
□ Show All Vertical Grid Lines

Status Line: None
□ Include Value

Max Column Width:
Max Row Label Width:

Figure 11.3: The **Properties** dialog box

Properties dialog box containing only those tabs that are common to all the selected objects. AIMMS will not display an initial value for the corresponding properties, as each property may hold different initial values for the various objects. Only the properties that you change are applied to the selected objects.

Property types

Through the tabs in the **Properties** dialog box, AIMMS lets you modify the various properties of pages and page objects. The following paragraphs provide a brief overview of the modifiable properties. A full explanation of the various properties of all the available objects can be found in the help file accompanying the AIMMS system.

The Contents tab

With the **Contents** tab you can add or remove identifiers from the list of identifiers that are displayed in the object. With this tab you can specify, for instance, that a table is to display the values of two or more identifiers. To modify the contents, AIMMS will open the common **Identifier Selection** dialog box as explained in Section 10.3.

Applying changes

Before you can make changes to the **Contents** tab, AIMMS requires that you apply any changes you have made to the other object properties before entering the **Contents** tab. You can apply these changes using the **Apply** button. Similarly, after you have made changes to the **Contents** tab, AIMMS requires that you apply these changes before you can go on to modify other object properties.

The Procedure tab

With the **Procedure** tab you can specify the particular procedures that must be executed upon user inputs such as a data change or selecting a particular value in a data object. The use of procedures linked to data objects is mostly to perform error checks or update other identifiers based on a single data change.

The Action tabs

With the **Action** tab, the counterpart of the **Procedure** tab for pages, buttons and navigational controls, you can specify the particular actions that must be executed upon opening a page, pressing a button, or making a selection in a navigational control. Such actions typically can be a sequence of running a procedure within the model, executing predefined AIMMS menu actions, or checking assertions.

The Menu tab

The **Menu** tab lets you specify which menu bar, toolbar, and right-mouse popup menu should be active on top of either a page or an object on a page. The menus themselves, as well as the actions linked to the menus, can be created in the **Menu Builder** tool. The **Menu Builder** tool is explained in full detail in Chapter 12.

Double-click actions

An action type that is used quite frequently, is the double-click action. You can specify a double-click action either in the **Action** tab, or through the **Menu** tab. The following rules apply.

- If a **Double-Click** procedure is specified on the **Action** tab, AIMMS will execute that procedure.
- If no **Double-Click** procedure has been specified, but a pop-up menu associated with the page object has a default item, AIMMS will execute the default menu item.
- If neither of the above apply, and the object is a table displaying a set, the double-click action will toggle set membership of the set element which currently has the focus.
- In all other cases, double-clicking will be ignored.

The Assert tab

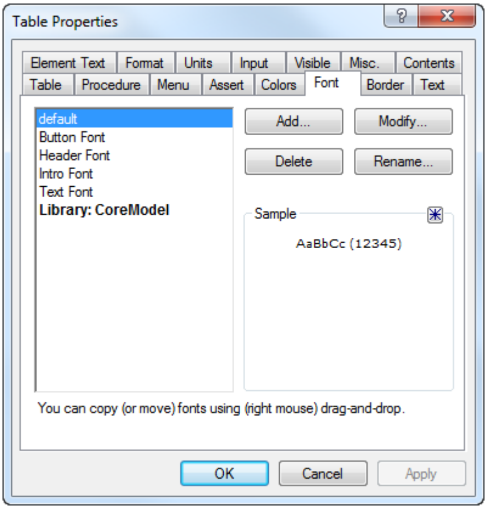
Through the **Assert** tab you can indicate which assertions already declared in your model are to be checked upon end-user data changes to a particular identifier in a data object. AIMMS can perform the assertion immediately upon every data change, or delay the verification until the end-user presses a button on the page. Once an immediate assertion fails, the assertion text will be displayed to the user and the original value will be restored.

The Colors tab

With the **Colors** tab you can not only specify the colors that are to be used for the foreground and background of a page or page object, but also the color for the user-selected values in a page object. In addition, you can specify a model-defined (indexed) color parameter to define the foreground color that will be used for each identifier in a data object. With such a parameter you can, for instance, color *individual* values of an object depending on a certain threshold. The necessary computations for this individual coloring need to be made inside the model underlying the end-user interface. You will find more details about assigning color parameters in Section 11.4.

The Font tab

The **Font** tab lets you define the font that is to be used for a particular object. You can choose the font from a list of user-defined font descriptions as illustrated in Figure 11.4. To add a new font name to the list, you should press the **Add** button. This will open the standard Windows font selection dialog box, allowing you to define a new AIMMS font based on the list of available Windows fonts, font styles and sizes. Once you have made a selection, you will be requested to provide a description for the newly selected font.



You can copy (or move) fonts using (right mouse) drag-and-drop.

Figure 11.4: The **Font** tab of a **Properties** dialog box

Choose functional font names

It is strongly recommended that you choose functional names for AIMMS fonts (i.e. describing their intended use) instead of merely describing the choices you made. For instance, naming a new font “Button font” instead of “Arial Regular, 8 pt” will help tremendously in preventing mistakes when selecting a font for a button.

Fonts in library projects

AIMMS also allows you to store fonts within a library project. The list of fonts shown in Figure 11.4 displays a single font *Small Table Font* associated with the library *CoreModel*. You can manage the list of fonts associated with a library by pressing the buttons on the right-hand side of the dialog box, while the selection in the listbox on the left-hand side is in the area associated with the library.

Use only in library pages

If you have defined fonts within a library project, you should ideally only use these fonts in pages that are also part of the library project. If you use the fonts in pages outside of the library, such pages may fail to display properly after you have removed the library project from the AIMMS project.

Font names must be unique

AIMMS requires that all font names be unique across the main project and all library projects that are included in the main project. If you include an existing library project, which contains a font name that is already present in the AIMMS project, AIMMS assumes that both fonts are the same and will ignore the second font definition.

The Border tab

With the **Border** tab you can stipulate the border settings for any particular data object on a page. A border can consist of merely a surrounding line, or provide an in- or out-of shadow effect.

The Text tab

With the **Text** tab you can specify for each identifier a single line of text that is displayed in a page object. With this line of text you can, for instance, provide descriptions for the data in a table containing one or more identifiers. In addition, the **Text** tab will let you define the element description for the (optional) status line associated with the object. The status line will display the currently selected value along with its element description. If the element description contains references to the indices over which the identifier at hand is defined, these references will be expanded to the currently selected element names.

The Element Text tab

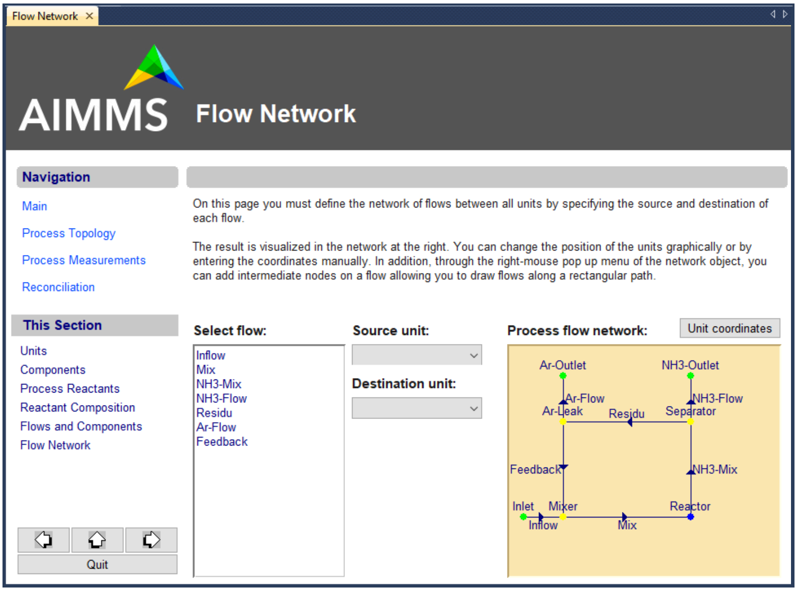
By default, any set element in a data object will be displayed by its name in the model. If you want to display an alternative text for a set element, you can use the **Element Text** tab to specify a string parameter holding these alternative element descriptions. You can use this feature, for instance, to display set elements with their long description in the end-user interface, whereas the model itself, and perhaps paper reports, work with short element names.

The Format tab

The **Format** tab defines the numerical format in which the data of a particular identifier is displayed. This format can be specified on the spot, or can use a named format already predefined by you as the application developer. The display format specifies not only such properties as the width of a number field and its number of decimal places, but also their relative alignment, the use of a 1000-separator for large numbers, and the display of default values.

The Units tab

The AIMMS modeling language offers advanced support for defining units of measurement for each identifier in the model. In particular, AIMMS supports unit conventions which let you define a coherent set of units (e.g. Imperial or metric units) in a single declaration. In the end-user interface you can indicate in the **Units** tab whether you want units to be displayed for every identifier or for every individual value contained in a particular data object. The displayed units are the units defined for the identifier at hand, unless the end-user has selected a current unit convention with alternative units. Figure 11.5 illustrates an end-user page in which identifier values are displayed along with their associated units of measurement.



On this page you must define the network of flows between all units by specifying the source and destination of each flow.

The result is visualized in the network at the right. You can change the position of the units graphically or by entering the coordinates manually. In addition, through the right-mouse pop up menu of the network object, you can add intermediate nodes on a flow allowing you to draw flows along a rectangular path.

Figure 11.5: Use of units in a data object

The Input tab

With the **Input** tab you can specify the read-only properties of every identifier in a page object separately. The decision as to whether numbers are read-only can depend on (indexed) identifiers in your model. Thus, you can arrange it so that particular numbers in, for example, a table can be edited by the end-user, while other numbers associated with that same identifier are considered as read-only. In addition to the properties specified on this tab, the overall read-only behavior of identifiers is also influenced by the contents of the predefined identifier `CurrentInputs` (see Section 17.1).

The Visible tab

You can use the **Visible** tab to hide a particular page object in its entirety from a page. Whether or not a page object is visible may depend on a scalar identifier (slice) in your model. The ability to hide page objects comes in handy when, for instance,

- you want to hide a page object because a particular end-user has no right to modify its data, or

- a page contains two exactly overlapping page objects—e.g. one holding relative numbers, the other holding absolute numbers—and you want to display just the one based on the user's choice.

The Misc. tab

With the **Misc.** tab you can specify various miscellaneous settings such as

- whether a page object must be included in the page tab order to specify a natural navigation order on the page (see also Section 11.1),
- whether an object is actually printed or skipped during printing (only relevant for print pages, see also Chapter 14),
- which end-user help topic should be displayed for the page or page object at hand, or
- a tag name, which is used when you want to refer to the object from within the model (see Section 17.3.1).

Help file

Before adding end-user help to a particular page, page object, end-user menu or toolbar, you must add a help file to your project directory, and specify its name through the **Options** dialog box (see Section 20.1). All the available end-user help associated with your project must be contained in the specified project help file.

Help file formats

Aimms supports several help file formats, allowing you to create a help file for your project using the tools you are most familiar with. They are:

- standard Windows help files (with the `.hlp` extension),
- compiled HTML help files (with the `.chm` extension), and
- PDF files (with the `.pdf` extension), which require that Acrobat Reader version 4.0 or higher is installed on your machine.

An executable Acrobat Reader installation can be downloaded from the Adobe website www.adobe.com.

Creating help files

To create a help file in any of the supported formats you will need an appropriate tool such as RoboHelp, Help & Manual or DocToHelp to create either a Windows or compiled HTML help file, or Adobe Acrobat to create a PDF file. To jump to a marked position inside the help file when providing help for a page, a page object, a menu or a button on a toolbar you should add:

- (so called) *K-keywords* to an ordinary Windows help file,
- *keywords* to a compiled HTML help file, or
- *named destinations* added to a PDF file.

All of the destinations that you added to the help in this way can serve as the **Help Topic** inside the **Misc.** tab of a page or page object.

Object-dependent properties

In addition to the tabs described above, which are common to most objects, the **Properties** dialog box also has a number of tabs where you can change properties that are very specific for a particular type of object. Through such object-dependent options you can specify, for instance, whether a table should display default values, what should be displayed along the axes in a graph or chart, or how the arcs and nodes in a network flow object should be drawn. The contents of these object-specific tabs are explained in full detail in the online AIMMS help file.

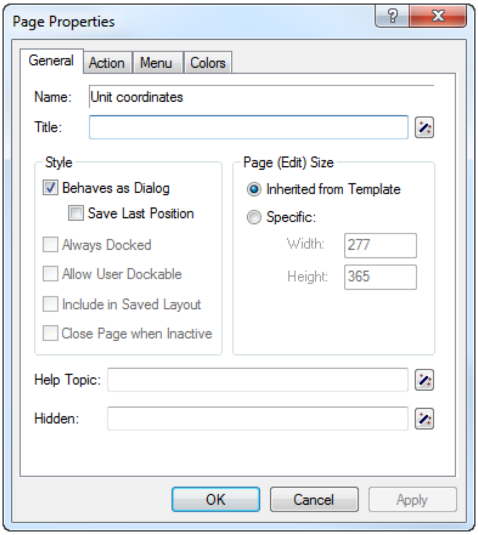
11.3 Using pages as dialog boxes

Use of dialog boxes

By default all end-user pages behave as normal windows, i.e. whenever you have multiple windows open, you can freely switch from window to window simply by clicking in the window that should become active. Sometimes, however, your application may contain sequential actions which require the user to make a certain choice or data change before moving on to the next action. In this case the page should behave as a dialog box instead of a normal window. While a dialog box is displayed on the screen, it is impossible to access other windows in the application without closing the dialog box first for example with an **OK** or **Cancel** button. By using dialog boxes you can force an end-user to follow a strict sequence of operations.

Dialog pages

In AIMMS you can define that a page should behave like a dialog box by using the page **Properties** dialog box as illustrated in Figure 11.6. If such a dialog



Name: Unit coordinates

Title:

Style

☑ Behaves as Dialog

□ Save Last Position

□ Always Docked

□ Allow User Dockable

□ Include in Saved Layout

□ Close Page when Inactive

Page (Edit) Size

● Inherited from Template

○ Specific:

Width: 277

Height: 365

Help Topic:

Hidden:

Figure 11.6: Creating a dialog page

page is opened using either a button, a menu, a navigation object or from within the model through a call to the PageOpen procedure, it will behave like a dialog box. *If, on the other hand, the dialog page is opened from within either the **Page Manager** or the **Template Manager**, the page will behave as an ordinary window.* This offers you the possibility of editing the contents and layout of the page.

Blocking execution

When a dialog page is called from within a procedure using PageOpen, the execution of the calling procedure will only continue after the dialog page has been closed by the end-user. In this way, any data supplied by the end-user in the dialog page will always be available during the remaining execution of the calling procedure.

Dialog box result

Note that dialog pages do not offer built-in support to determine whether an end-user has finished the dialog box for example by pressing the **OK** or **Cancel** button. However, such control can easily be modeled in the AIMMS language itself. Perhaps the most straightforward manner to accomplish this is by introducing

- a set DialogActions containing two elements 'OK' and 'Cancel',
- an associated global element parameter CurrentDialogAction, and
- procedures such as ButtonOK and ButtonCancel which set CurrentDialogAction equal to 'OK' or 'Cancel', respectively.

Linking to dialog box buttons

To obtain the result of a dialog page, you can simply add the execution of the procedures ButtonOK or ButtonCancel to the list of actions associated with the **OK** and **Cancel** buttons, respectively. In addition, you should link the functionality of the close icon for the dialog page to that of the **Cancel** button as illustrated in Figure 11.7.

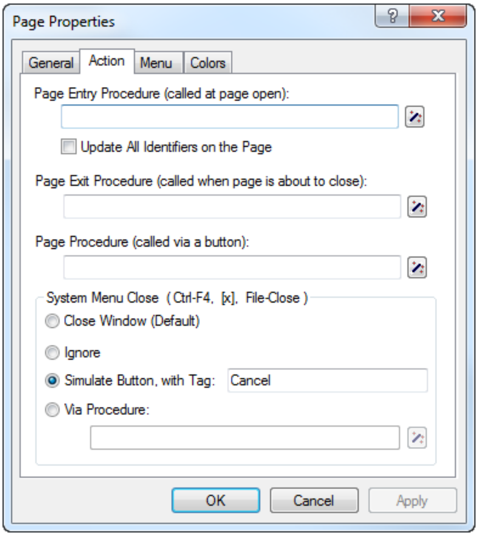
Obtaining the result

To obtain the end-user choice in the dialog page after the return of the PageOpen procedure, you can simply check for the value of the element parameter CurrentDialogAction, as illustrated in the following code excerpt.

```
! Open the dialog page and stop processing when the user
! has pressed the 'Cancel' button.

OpenPage( "Supply input data" );
return 0 when CurrentDialogAction = 'Cancel';

! Otherwise perform further data processing based on the supplied input data.
```

Create a dialog page template

You may want to create a customized dialog page *template* (see also Section 12.2) to capture the end-user choices as described above. Based on such a dialog page template, you can quickly create as many dialog pages as necessary, all behaving in a similar fashion when opened in a procedure of your model.



Page Properties

General | Action | Menu | Colors

Page Entry Procedure (called at page open):

□ Update All Identifiers on the Page

Page Exit Procedure (called when page is about to close):

Page Procedure (called via a button):

System Menu Close (Ctrl-F4, [x], File-Close)

○ Close Window (Default)

○ Ignore

● Simulate Button, with Tag: Cancel

○ Via Procedure:

Figure 11.7: Linking dialog close to **Cancel**

11.4 Defining user colors

User colors

As already explained in the previous section, AIMMS allows you to define the color of particular objects in a graphical end-user interface from within the execution of your model. In this section you will see how you can define *user colors* which can be used within the model, and how you can use them to provide model-computed coloring of page objects.

Defining persistent user colors

To define user colors that persist across multiple sessions of a project, you should open the **User Colors** dialog box as illustrated in Figure 11.8 from the **Tools** menu. By pressing the **Add** or **Change Color** button, AIMMS will display the standard Windows color selection dialog box, which you can use to create a new user color or modify an existing user color. After you have selected a color, AIMMS will request a name for the newly defined color for further usage within the model.

Functional color names

As with font names, you may prefer to choose functional color names rather than names describing user colors. For instance, colors named “Full tank color”, “Partially filled color” and “Empty tank color” may be a much better choice, from a maintenance point-of-view, than such simple names as “Red”, “Blue” and “Green”. In addition, choosing descriptive names may make the intention of any assignment to, or definition of, color parameters in your model much clearer.

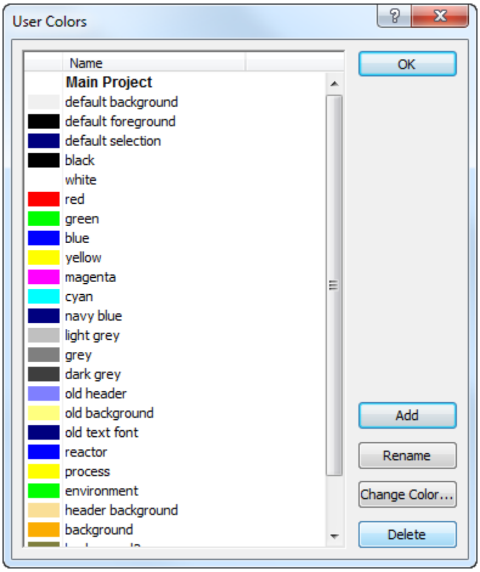


Figure 11.8: The **User Colors** dialog box

User colors in library projects

Similar as with fonts, a library project can also contain its own set of user colors. The list of colors shown in Figure 11.8 displays the user colors defined within the main project. For each library included in the project the listbox contains a separate area displaying the user colors that are associated with that library. You can manage the list of user colors associated with a library by pressing the buttons on the right-hand side of the dialog box, while the selection in the listbox on the left-hand side is in the area associated with the library.

Use only in library pages

If you have defined user colors within a library project, you should ideally only use these user colors in pages that are also part of the library project. If you use the user colors in pages outside of the library, such pages may fail to display properly after you have removed the library project from the AIMMS project.

Color names must be unique

AIMMS requires that all user color names be unique across the main project and all library projects that are included in the main project. If you include an existing library project, which contains a user color name that is already present in the AIMMS project, AIMMS assumes that both user colors are the same and will ignore the second color definition.

Runtime user colors

Persistent user colors cannot be modified or deleted programmatically. However, you can add runtime colors (which only exist for the duration of a project session) programmatically from within your model using the function UserColorAdd. In the **User Colors** dialog box, runtime colors are shown under the header **Runtime colors**. You can modify or delete such runtime colors using the functions UserColorModify and UserColorDelete. These functions are discussed in full detail in Section 17.2.1.

The set AllColors

All persistent and non-persistent user colors are available in your model as elements of the predefined set AllColors. To work with colors in your model you can simply define scalar and/or indexed element parameters into the set AllColors. Through simple assignments or definitions to such parameters you can influence the coloring of identifiers or individual identifier values on an end-user page.

Example

Consider a set of Flows in a network with index f. If a mathematical program minimizes the errors in computed flows in respect to a set of measured flow values, then the following simple assignment to a color parameter FlowColor(f) marks all flows for which the error exceeds a certain threshold with an appropriate color.

```
FlowColor(f) := if ( FlowError(f) >= ErrorThreshold ) then
    ’Red’ else ’Black’ endif;
```

Use in interface

With the above assignment, any graphical display of Flows can be colored individually according to the above assignment by specifying that the color of the individual numbers or flows in the **Colors** dialog box of the object be given by the value of the color parameter FlowColor(f). Figure 11.5 (on page 143) illustrates an example of an end-user page where the flows in the network flow object, as well as the individual entries in the tables and lists, are colored individually with respect to the parameter FlowColor(f) (the colors are only visible in the electronic version of this book).

Chapter 12

Page Management Tools

This chapter

When your decision support system grows larger, with possibly several people developing it, its maintainability aspects become of the utmost importance. To support you and your co-workers in this task, AIMMS offers several advanced tools. As discussed in Chapters 4 and 7, the **Model Explorer** combined with the **Identifier Selector** and **View Manager**, provide you with various useful views of the model's source code. In this chapter, the specialized AIMMS tools that will help you set up an advanced end-user interface in an easily maintainable manner will be introduced.

12.1 The Page Manager

Page navigation

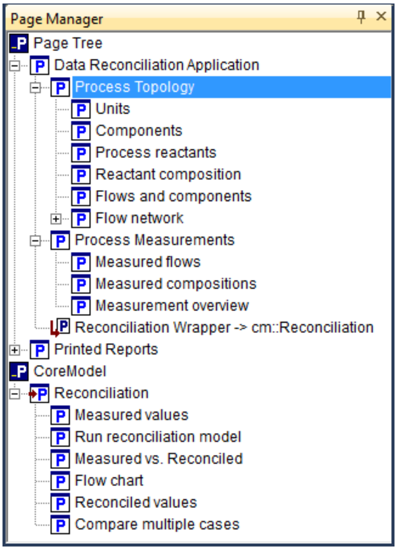
In large decision support systems with many pages, navigating your end-users in a consistent manner through all the end-user screens is an important part of setting up your application. One can think of several organizational structures for all the available end-user screens in your application that would help your end-users maintain a good overview of their position. To help you set up, and modify, a clear navigational organization quickly and easily, AIMMS provides a tool called the **Page Manager**.

The Page Manager

With the **Page Manager** you can organize all the existing pages of an AIMMS application in a tree-like fashion, as illustrated in Figure 12.1. The tree in the **Page Manager** that holds all the pages of the main AIMMS project is called the main *page tree*. Relative to a particular page in this page tree, the positions of the other pages define common page relationships such as *parent* page, *child* page, *next* page or *previous* page.

Library page trees

In addition to the main page tree, each library project included in the main project can have its own tree of pages as illustrated in Figure 12.1. Section 12.1.1 discusses the facilities available in AIMMS that allow you to combine the page structures in all trees to construct a single navigational structure for the entire application.


Page Manager

- Page Tree
  - Data Reconciliation Application
    - Process Topology
      - Units
      - Components
      - Process reactants
      - Reactant composition
      - Flows and components
      - Flow network
    - Process Measurements
      - Measured flows
      - Measured compositions
      - Measurement overview
    - Reconciliation Wrapper -> cm::Reconciliation
  - Printed Reports
- CoreModel
  - Reconciliation
    - Measured values
    - Run reconciliation model
    - Measured vs. Reconciled
    - Flow chart
    - Reconciled values
    - Compare multiple cases

Figure 12.1: The **Page Manager**

Navigational controls

The page relationships defined by the page tree can be used in several navigational interface components that can be added to a page or end-user menu. These components include

- *navigation objects*,
- *navigation menus*,
- *button actions*, and
- *tabbed pages*.

These allow you to add dynamic navigation to the parent, child, next or previous pages with respect to the position of either

- the current page, or
- a fixed page in the page tree.

Section 12.1.2 explains in detail how to set up such automatic navigation aids.

Aimed at ease of maintenance

The strength of the **Page Manager** tool lies in the fact that it allows you to quickly add pages to the page tree, delete pages from it, or modify the order of navigation without the need to make modifications to hard-coded page links on the pages themselves. Thus, when a model extension requires a new section of pages, you only need to construct these pages, and store them at the appropriate position in the page tree. With the appropriate navigational interface components added to the parent page, the new page section will be available to the end-user immediately without any modification of existing pages.

12.1.1 Pages in library projects

Pages in library projects

AIMMS allows you to develop a separate page tree for every library project included in an AIMMS application. This is an important feature of library projects because

- it allows a developer to implement a fully functional end-user interface associated with a specific sub-project of the overall application completely independently of the main project, and
- pages defined inside a library project can refer to all the identifiers declared in that library, whereas pages defined in the main project (or in any other library) can only refer to the public identifiers in the interface of that library (see Section 3.2).

Duplicate names may occur

While AIMMS requires that the names of pages within a single (library) project be unique, page names need not be unique across library projects. To ensure global uniqueness of page names in the overall application, AIMMS internally prefixes the names of all the pages contained within a library with its associated library prefix (see Section 3.1). When you want to open an end-user page programmatically, for instance through the `PageOpen` function, you need to provide the full page name including its library prefix. Without a library prefix, AIMMS will only search for the page in the main page tree.

Separate page trees

The page trees associated with the main project and with all of its library projects are initially completely separate. That is, any navigational control (see Section 12.1.2) that refers to parent, child, next or previous pages can never navigate to a page that is not part of the tree in which the originating page was included.

All pages are accessible

Other than for the identifier declarations in a libray, AIMMS puts no restriction on which pages in the library can and cannot be shown from within the main application, or from within other libraries. Stated differently, the page tree of a library does not currently have a public interface.

Creating an application GUI

If an AIMMS project is composed of multiple libraries, then each of these libraries contains its own separate page tree, which may be combined to form the total end-user interface of the overall application. The navigational controls offered by AIMMS, however, can only reach the pages in the same tree in which an originating page is included.

Jumping to library pages

Without further measures, pages from different libraries would, therefore, only be accessible through a unidirectional direct link, which is very undesirable from an end-user perspective. After following such a link moving to a parent, next or previous page may give completely unexpected results, and getting back to the originating page may be virtually impossible. For both developers and end-users a situation in which all relevant pages can be reached from within a single navigational structure is much more convenient.

Page tree references

To address this problem, AIMMS offers a linkage mechanism called *page tree references*. Through a page tree reference, you can *virtually* move a subtree of pages from a library project to another location in either the main project or any other library project included in the AIMMS application. While physically the pages remain at their original location, the navigational controls in AIMMS will act as if the tree of pages has been moved to the destination location of the page tree reference. At the original location AIMMS' navigational controls will completely disregard the linked page tree.

Creating a page tree reference

You can create a page tree reference by inserting a page tree reference node at the destination location through the **Edit-New-Page Tree Reference** menu. In figure 12.1 the *Reconciliation Wrapper* node illustrates an example of a page tree reference node. It is linked to the tree of pages starting at the *Reconciliation* page in the *CoreModel* library. Note that AIMMS uses a special overlay of the page icon to visualize that a page is linked to a page tree reference node, and hence, at its original location, is not reachable anymore through AIMMS' navigational controls.

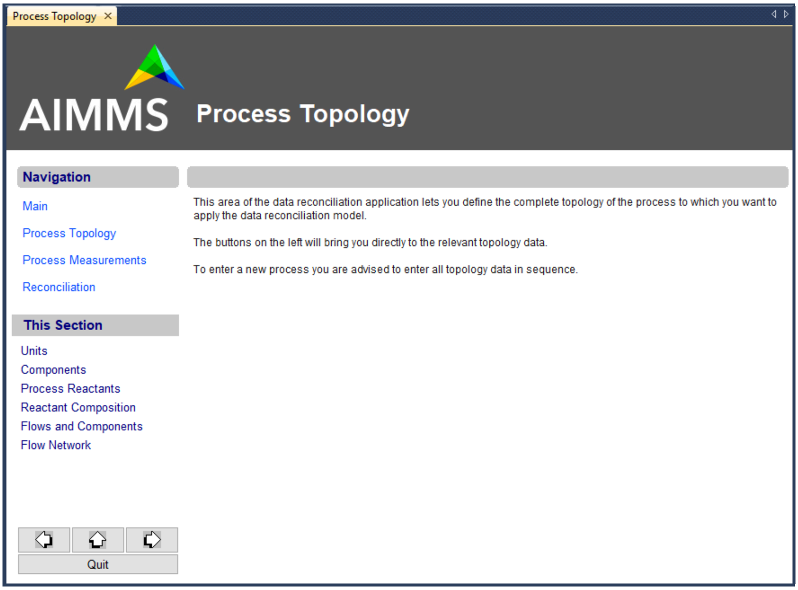
Linking a page tree reference

To create a link between a page tree reference node and a subtree of pages anywhere else in your application you have to select both the page tree reference node and the node that is at the root of the subtree that you want to link to, and select the **Edit-Page Tree Reference-Link** menu. You can unlink an existing link through the **Edit-Page Tree Reference-Unlink** menu.

12.1.2 Navigational interface components

Navigational control

The page tree can be used to directly control the navigational structure within an AIMMS-based end-user application. This can be accomplished either by special button actions or through the navigation object and menus. As an example, Figure 12.2 illustrates the *Process Topology* page contained in the page tree of Figure 12.1. In the lower left corner, the page contains three navigational buttons that are linked, from left to right, to the previous, parent and next page. Above this, the page contains a navigation object which, in this instance, automatically displays a list of buttons that corresponds exactly to the set of direct child nodes of the *Process Topology* page in the page tree.



This area of the data reconciliation application lets you define the complete topology of the process to which you want to apply the data reconciliation model.

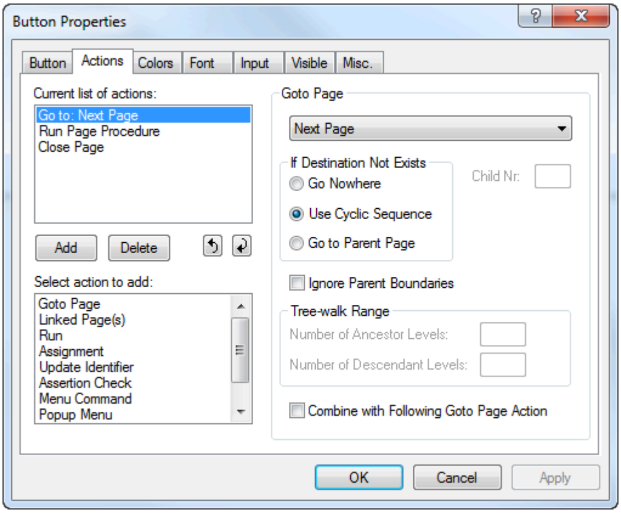
The buttons on the left will bring you directly to the relevant topology data.

To enter a new process you are advised to enter all topology data in sequence.

Figure 12.2: Page containing navigation buttons and a navigation object

Button actions

To add a page tree-based navigational control to a button, you only need to add a **Goto Page** action to the **Actions** tab in the button **Properties** dialog box, as illustrated in Figure 12.3. You can request AIMMS to open the previous,



Current list of actions:
Go to: Next Page
Run Page Procedure
Close Page

Select action to add:
Goto Page
Linked Page(s)
Run
Assignment
Update Identifier
Assertion Check
Menu Command
Popup Menu

Goto Page: Next Page

If Destination Not Exists
○ Go Nowhere
● Use Cyclic Sequence
○ Go to Parent Page

Child Nr:

□ Ignore Parent Boundaries

Tree-walk Range
Number of Ancestor Levels:
Number of Descendant Levels:

□ Combine with Following Goto Page Action

Figure 12.3: Adding navigational control to a button

next, parent or (first) child page relative to the position of the current page in the page tree. If you want the current page to be closed after opening the new page, you should additionally insert a **Close Page** action as in Figure 12.3.

Cycling

When there is no longer a next or previous page to open in a particular branch of a page tree, AIMMS will cycle to the first or last page within that branch, respectively. You can further modify the result of a previous or next page action by placing special *separator* nodes into the page tree, given that AIMMS will never jump past a separator node. You will find the full details of separator nodes in the online help on the **Page Manager**.

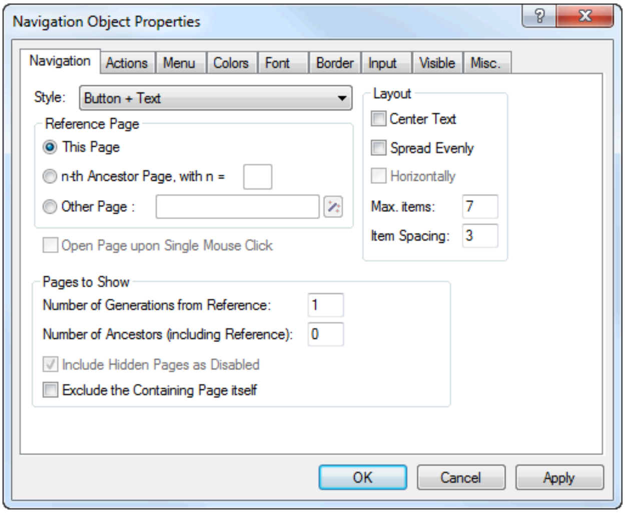
Navigation object

The second way to include a navigational control in an end-user page is by means of a custom *navigation* object. A navigation object can display a subtree of the entire page tree in several formats, such as:

- a list of buttons containing the page titles (as in Figure 12.2),
- a list of buttons accompanied by the page titles,
- a list of clickable or non-clickable page titles without buttons, or
- a tree display similar to the page tree itself.

Object properties

After adding a navigation object to a page, you must specify the subtree to be displayed through the **Properties** dialog box as displayed in Figure 12.4. What is displayed in the navigation object is completely determined by the reference page, together with the number of ancestor (parent) and child generations specified in this dialog box.



Style: Button + Text

Reference Page

● This Page

○ n-th Ancestor Page, with n =

○ Other Page :

□ Open Page upon Single Mouse Click

Layout

□ Center Text

□ Spread Evenly

□ Horizontally

Max. items: 7

Item Spacing: 3

Pages to Show

Number of Generations from Reference: 1

Number of Ancestors (including Reference): 0

☑ Include Hidden Pages as Disabled

□ Exclude the Containing Page itself

Figure 12.4: Navigation object **Properties** dialog box

Display only

If you set a navigation object to read-only using the **Input** tab of the **Properties** dialog box, then you can use the navigation object for display- only purposes. Thus, you can use it to display the current page title as a page header, or the title of one or more parent pages in the header or footer area of the page. The "*Process Topology*" page header of the end-user page displayed in Figure 12.2 is an example of a display-only navigation object.

Navigation menus

Finally, you can add special navigation (sub)menus to your application in which the menu items and submenus represent a subtree structure of the page tree. Figure 12.5 illustrates an example of a navigation menu linked to the page tree displayed in Figure 12.1.

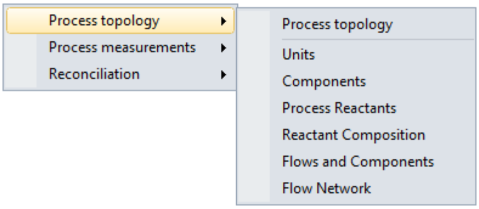


Figure 12.5: Example of a navigation menu

Adding navigation menus

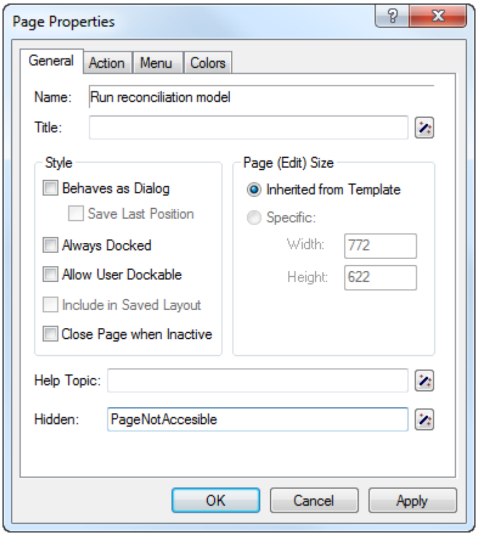
You can add a navigation menu to any menu in the **Menu Builder** tool (see Section 12.3). For each navigation menu you must specify a reference page and the scope of the subtree to be displayed in a similar fashion to that illustrated for the navigation object in Figure 12.4.

Hiding pages

Pages can be statically or dynamically hidden using the page **Properties** dialog box (see also Section 11.2), as illustrated in Figure 12.6. In the **Hidden** field, you must enter a scalar value, identifier or identifier slice. Whenever the property assumes a nonzero value the page is hidden, and automatically removed from any navigational interface component in which it would otherwise be included.

Authorizing access

For larger applications, end-users can usually be divided into groups of users with different levels of authorization within the application. Disabling pages based on the level of authorization of the user (explained in Chapter 19) then provides a perfect means of preventing users from accessing those data to which they should not have access. You can still open a hidden page via a hard-coded page link.



Page Properties

General | Action | Menu | Colors

Name: Run reconciliation model

Title:

Style

□ Behaves as Dialog

□ Save Last Position

□ Always Docked

□ Allow User Dockable

□ Include in Saved Layout

□ Close Page when Inactive

Page (Edit) Size

● Inherited from Template

○ Specific:

Width: 772

Height: 622

Help Topic:

Hidden: PageNotAccesible

Figure 12.6: Hiding a page

12.2 The Template Manager

Consistent look-and-feel

Complementary to the **Page Manager** is the AIMMS **Template Manager**. Using the **Template Manager**, you can ensure that all pages are the same size and possess the same look-and-feel, simply by positioning all end-user pages in the template tree associated with a project. An example of a template tree containing both templates and end-user pages is displayed in Figure 12.7.

Hierarchical template structure

In addition to all the end-user pages, the template tree can contain a hierarchical structure of template pages. Within the template tree, template pages behave as ordinary pages, but they are not available to end-users. Through templates you can define common page objects that are shared by all the template *and* end-user pages positioned below a particular template in the template tree.

Duplicating page templates

When you want to use the same template page at two or more distinct positions in the template tree, AIMMS lets you duplicate, rather than copy, the template node containing that component. Changes made to the duplicated page template at any position in the template tree, are automatically propagated to all other occurrences. Duplicated templates can be recognized by the duplication symbol ⧉ which is added to the icon of every duplicate template in the template tree.

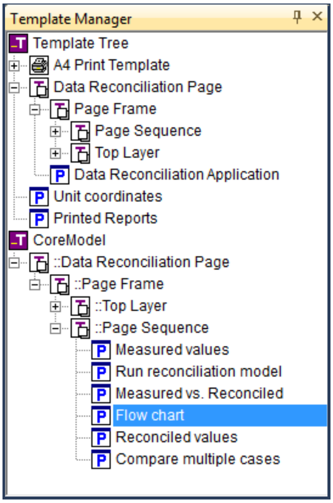


Figure 12.7: The **Template Manager**

End-user pages automatically added

Every new end-user page created in the **Page Manager**, is automatically added to the root node in the template tree. By moving the page around in the template tree, it will inherit the combined look-and-feel of all templates above it.

Common page components

The hierarchical structure of the template tree lets you define layers of common objects on top of each other. Thus, a first template might globally define the page size and background color of all underlying pages, while a second template could define common components such as a uniformly shaped header and footer areas. As an example, Figure 12.8 illustrates a template for an end-user page from the template tree of Figure 12.7, in which the components defined in various templates are identified.

Modify look-and-feel

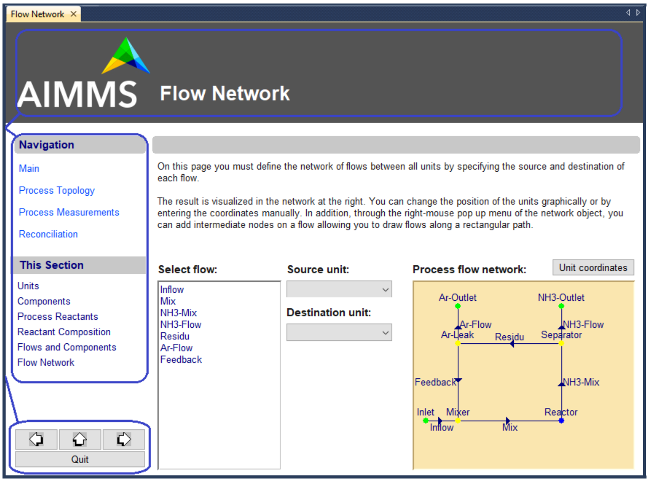
You can quickly modify the entire look-and-feel of your application, by moving a subtree of templates and end-user pages from one node in the template tree to another. Thus, the entire look-and-feel of page size, header and footer areas, background color and navigational area(s) of all pages in an Aimms application can be changed by a single action.

Template objects not editable

When you open a template or end-user page in the template manager, it will be opened in edit mode by default, and inherit all the properties of, and all objects contained in, the templates above. On any template or end-user page you can only modify those objects or properties that are defined on the page itself. To modify objects defined on a template, you must go to that template and modify the objects there.

Inherited from *Page Frame*

Inherited from *Page Sequence*



On this page you must define the network of flows between all units by specifying the source and destination of each flow.

The result is visualized in the network at the right. You can change the position of the units graphically or by entering the coordinates manually. In addition, through the right-mouse pop up menu of the network object, you can add intermediate nodes on a flow allowing you to draw flows along a rectangular path.

Select flow: Inflow, Mix, NH3-Mix, NH3-Flow, Residu, Ar-Flow, Feedback

Source unit:

Destination unit:

Process flow network:

Figure 12.8: Example of an end-user page using templates

Combine with navigational components

You can achieve an exceptionally powerful combination by adding navigational components to a template page. If the reference page property of such a navigational component is expressed in terms of the current page, or one of its ancestor pages, then, in end-user mode, the current page will always refer to the particular end-user page which uses that template. Thus, given a well-structured page tree, you potentially only need a *single* template to add navigational control components to *all* end-user pages. This is particularly true for such common controls as **Previous** and **Next** buttons.

12.2.1 Templates in library projects

Templates in libraries

Each library project in AIMMS has a separate template tree, which is available as a separate root node in the **Template Manager**, as illustrated in Figure 12.7. Pages in a library must be positioned in the template tree of that library to obtain their look-and-feel. This allows the developer of a library project to define the look-and-feel of the pages in the library completely independent of the main project and other library projects.

Sharing templates with the main project

If you want the pages of an entire application to share a common look-and-feel across all library projects included in the application, AIMMS also allows you to duplicate template pages from the main project into a library project. Thus, any changes made to the templates in the main project are automatically inherited by the pages in the library that depend on the duplicates of these templates.

Sharing templates across multiple projects

Conversely, you can also use library projects to enable the end-user GUIs of multiple AIMMS project to share a common look-and-feel. By defining templates, fonts, and colors in a single library project, and including this library project into multiple AIMMS projects, the pages in these projects can depend on a single, shared, collection of page templates. Thus, changes in a single template library will propagate to all AIMMS projects that depend on it.

Moving pages to a library

If you move or copy pages from the main project to a library project (or between library projects), AIMMS will automatically duplicate the relevant template structure from the source project to the destination project. This ensures that the pages have the exact same look-and-feel at the destination location as they had at their source location.

Example

The automatic duplication behavior of AIMMS is illustrated by the page tree in Figure 12.1 and the template tree in Figure 12.7. These trees were created by moving the *Reconciliation* page and its child pages from the main project to the *CoreModel* library. Subsequently, AIMMS automatically duplicated the template structure in the *CoreModel* library to ensure the identical look-and-feel for the moved pages.

12.3 The Menu Builder

The Menu Builder

The last page-related design tool available in AIMMS is the **Menu Builder**. 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. The **Menu Builder** window is illustrated in Figure 12.9. In the **Menu Builder** window you can define menus and toolbars in a tree-like structure in a similar fashion to the other page-related tools. The menu tree closely resembles the natural hierarchical structure of menus, submenus and menu items.

Default menu bar and toolbar

As illustrated in Figure 12.9, the **Menu Builder** will always display four default nodes. Two nodes representing the standard end-user menu bar and toolbar. These bars are linked to all end-user pages by default. And two nodes representing the standard Data Manager menu bar and toolbar. Although non-editable, you can use these nodes to copy (or duplicate) standard end-user menus or submenus into your own customized menu bars and toolbars. The data manager items will allow you to build your own menu and toolbar for the data manager with the same functionality as the standard menu and toolbar for the data manager.

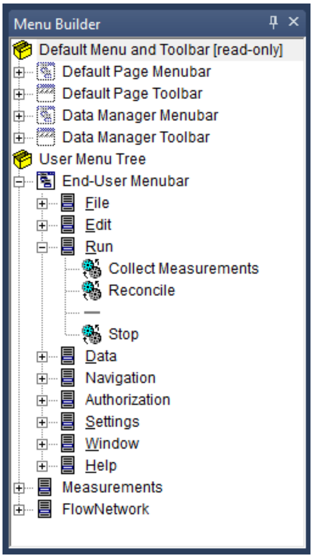


Figure 12.9: The **Menu Builder** window

Inserting new nodes

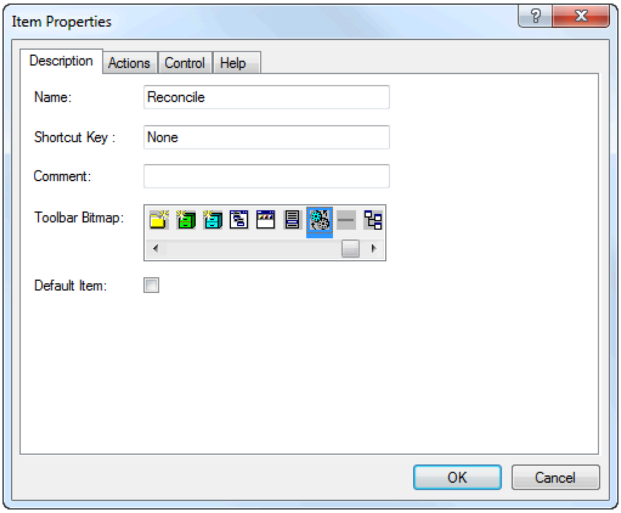
In the *User Menu Tree*, you can add nodes to represent menu bars, (sub)menus, menu items or toolbars in exactly the same manner as in other trees such as the model and page trees. Also, you can copy, duplicate or move existing nodes within the tree in the usual manner (see Section 4.3). The names given to menu and menu item nodes are the names that will be displayed in the end-user menus, unless you have provided a model-specific menu description in the menu **Properties** dialog box (e.g. to support multiple languages).

Menu item properties

For every node in the menu tree you can modify its properties through the **Properties** dialog box. In the **Properties** dialog box you can perform tasks such as linking end-user actions or model procedures to a menu item, provide shortcut keys, tooltips and help, or link a menu item to model identifiers that specify whether the item should be disabled within an end-user menu, or even be completely hidden from it. The **Properties** dialog box for a menu item is shown in Figure 12.10.

Adding menu actions

Through the **Actions** tab of the **Properties** dialog box, you can associate a list of actions with a menu item. Such actions can consist of executing menu items from system menus, navigational commands such as opening or closing pages, and also running procedures from your model, verifying assertions or updating identifiers.



Name: Reconcile

Shortcut Key : None

Comment:

Toolbar Bitmap:

Default Item: □

Figure 12.10: The menu item **Properties** dialog box

Hiding and disabling items

With the **Control** tab it is possible to provide control over a menu item from within your model. You can specify scalar 0-1 identifiers from within your model to determine whether a menu item or submenu should be disabled (grayed out) or completely hidden from the menu. Thus, you can prevent an end-user from performing tasks for which he is not authorized. In addition, you can couple a 0-1 identifier to a menu item in order to determine whether a menu item is checked, and which conversely toggles its value when an end-user checks or unchecks the item.

Tooltips and help

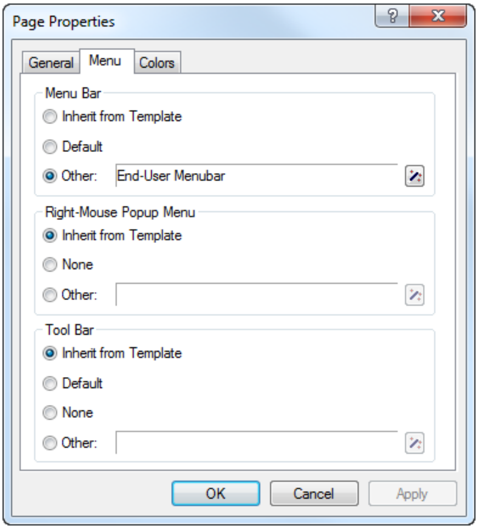
In the **Help** tab of the **Properties** dialog box, you can provide a description and help describing the functionality of a menu command. It lets you specify such things as the tooltips to be displayed for buttons on the button bar, a descriptive text for to be shown in the status bar, and a link to a help item in the project related help file.

Navigation menus

Navigation menus are a special type of menu that can be added to the menu tree. Navigation menus expand to a number of items in the current menu, or to one or more submenus, according to the structure of a particular subtree of the page tree as specified by you. Through navigation menus you can quickly and easily create menus that help an end-user navigate through your application. For example, you could create a menu item which links to the first child page, or to the parent page, of any page to which the menu is linked. The details of how to specify which pages are displayed in a navigation menu can be found in Section 12.1.2.

Linking to pages and objects

You can link a single menu bar, toolbar and pop-up menu to any end-user or template page in your project through the **Menu** tab of the page **Properties** dialog box, as illustrated in Figure 12.11 For every field in the dialog box, AIMMS lets you select an existing node in the menu tree. If you do not specify a menu bar or toolbar, AIMMS will automatically open the default end-user menu bar and toolbar.



Menu Bar

○ Inherit from Template

○ Default

● Other: End-User Menubar

Right-Mouse Popup Menu

● Inherit from Template

○ None

○ Other:

Tool Bar

● Inherit from Template

○ Default

○ None

○ Other:

Figure 12.11: Linking menus to pages

Inherited menus

When you add a menu bar or toolbar to a page template, these bars are automatically inherited by all pages that use that template. In this manner, you can quickly add your own customized end-user menu to all, or groups of, pages in your application. All new end-user pages will, by default, inherit their menu bar and toolbar from their templates.

12.3.1 Menus in library projects

Menus in library projects

In addition to the main *User Menu Tree* in the **Menu Builder**, each library project in AIMMS has a separate menu tree, as illustrated in Figure 12.9. In this menu tree, you can create the menus and toolbars that are specific for the pages defined in the library at hand.

Accessing private identifiers

When you are specifying the actions associated with a menu item in the menu tree of a library, you have access to all the identifiers and procedures declared in the library module of that library. For menu items in all other menu trees, you can only access the identifiers in the interface of the library.

Creating menus

When creating menus and toolbars in a library, you can duplicate menus and menu items from any other menu tree in the **Menu Builder**. Likewise, you can duplicate menus and menu items from a library project into the menu tree of the main project. This enables you to compose global menu- and toolbars that can be used in the overall application, yet containing library-specific submenus and menu items to dispatch specific tasks to the appropriate libraries.

Assigning menus to pages

When you want to assign a menu to a page or template, Aimms allows you to choose a user menu of either the main project or of any of its library projects. You should note, however, that choosing a menu from a different library project creates an implicit dependency on that project which is not immediately apparent in the page or template tree. If you copy or move pages with a user menu or toolbar from one project to another, Aimms will not duplicate that menu or toolbar, but still refer to their original locations as expected.

Chapter 13

Page Resizability

Resizability

Due to the diversity of objects and their position on a page, it is not immediately clear how objects should adjust when the size of a page is changed. Should buttons remain the same, when the size of particular data objects are changed? Such decisions are up to you, the developer of the application.

This chapter

In this chapter, you will learn about the facilities in AIMMS which you can use to specify how page components should scale when a page size changes. Such facilities allow you to create *resizable pages* which are ready for use with different screen resolutions. In addition, resizable pages let an end-user temporarily enlarge or reduce the size of a particular page to view more data on the same page, or to simultaneously look at data on another end-user page.

13.1 Page resizability

Choosing a base resolution

When you are developing an end-user interface around an AIMMS-based application for a large group of end-users, you must decide about the base screen resolution on which the end-user interface is intended to be run primarily. Such a decision is based on your expectations about the screen resolution that most of your end-user will be using. Fortunately, there is a tendency towards high-resolution screens amongst users.

Supporting different resolutions

Nevertheless, it is likely that one or more of your end-users will request to run the application at a different resolution. One reason could be that they use a notebook which does not support the base resolution you selected. Another reason could be that some of your end-users are working with such large data sets that a higher resolution would help them to have a better overview of their data.

Resizable pages

To help you support the demands of your end-users, AIMMS provides a fairly straightforward facility to create resizable pages and page templates. As you will see later on, the position and type of so-called *split lines*, placed on a resizable page, determines the manner in which objects on the page will scale upon resizing the page.

Non-resizable behavior

When a page has not been made resizable, all objects on that page will remain in their original position. Whenever such a page is reduced, and a data object included on the page falls outside the visible page area, AIMMS will automatically add horizontal or vertical scroll bars. If the page is increased in size, the bottom and right parts of the page will remain empty.

Adding resizability

To make a page resizable, the page should be put into **Resize Edit** mode, which is available in the **View-Resize Edit** menu of any page that is already in **Edit** mode. **Resize Edit** mode will replace all objects on the page by shaded rectangles. Figure 13.1 illustrates the **Resize Edit** view of the end-user page shown in Figure 12.8.

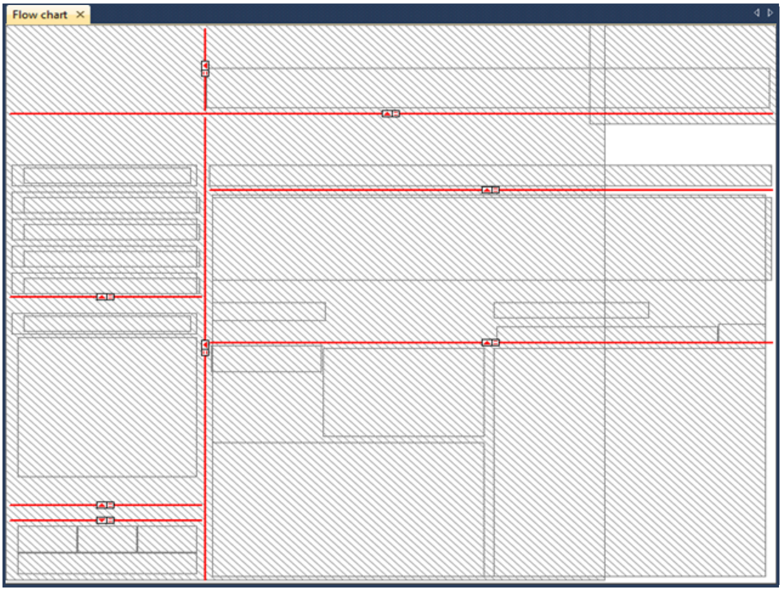


Figure 13.1: Page 12.8 in **Resize Edit** Mode

Split lines

A page is resizable as soon as it contains one or more horizontal or vertical *split lines*. The page in Figure 13.1 illustrates a number of such split lines. Each split line divides the rectangle in which it has been placed into two subrectangles. For each vertical split line you can decide either

- to keep the width of the left or right subrectangle constant (indicated by ◀ and ▶ markers),
- to ensure that the widths of the left and right subrectangles have the same ratio (indicated by ◀▶ marker), or
- to make the ratio between these two widths user-adjustable (indicated by ◁ or ▷ markers).

Similarly, horizontal split lines are used to indicate the relative height of the lower and upper subrectangles. On an end-user page, a user-adjustable split line will be visible as a split bar, which can be dragged to (simultaneously) resize the areas on both sides of the bar.

Stacking split lines

By selecting a subrectangle created by a split line, you can recursively subdivide that rectangle into further subrectangles using either horizontal or vertical split lines. What results is a specification of how every part of the page will behave relative to its surrounding rectangles if the size of the entire page is changed.

Adding split lines

One way of adding split lines to any subrectangle on a page in **Page Resize** mode is to select that subrectangle on the page (by clicking on it), and add a horizontal or vertical split line to it using one of the buttons from the **Page Resize** toolbar. Alternatively, if you want to insert a split line within an existing hierarchy of split lines, select the line just above where you want to insert a split line, and use one of the buttons to insert a new split line of the desired type.

Resize Try mode

By putting a page that is already in **Resize Edit** mode into **Resize Try** mode (via the **View-Resize Try** menu) and resizing the page, AIMMS will display the shapes of all page objects according to the specified resize behavior. Figure 13.2 illustrates the effect of resizing the page displayed in Figure 13.1 to a smaller size. These resized shapes are determined by calculating the new relative positions of all four corner points of an object within their re-

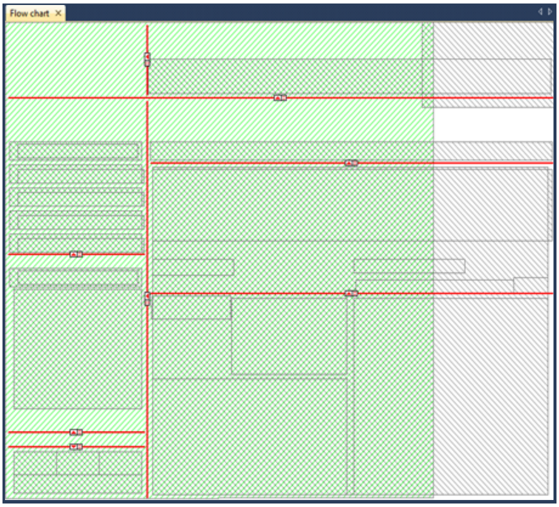
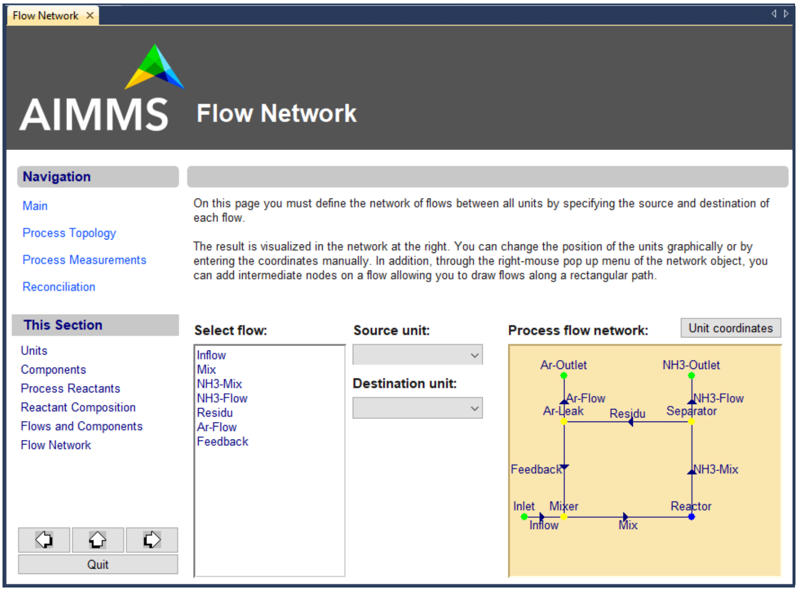


Figure 13.2: Resizing a resizable page

spective surrounding rectangles. This may result in nonrectangular shapes for some page objects, which are marked red. In such a situation, you should reconsider the placement of objects and split lines. Non- rectangularly shaped objects may distort the spacing between objects in end-user mode, because AIMMS will enforce rectangular shapes in end-user mode by only considering the top-left and bottom-right corners of every object.

Example

Consider the configuration of split lines illustrated in Figure 13.1, and its associated end-user page displayed in Figure 13.3. As already indicated in Figure 13.2, the particular combination of split lines results in the following behavior.



On this page you must define the network of flows between all units by specifying the source and destination of each flow.

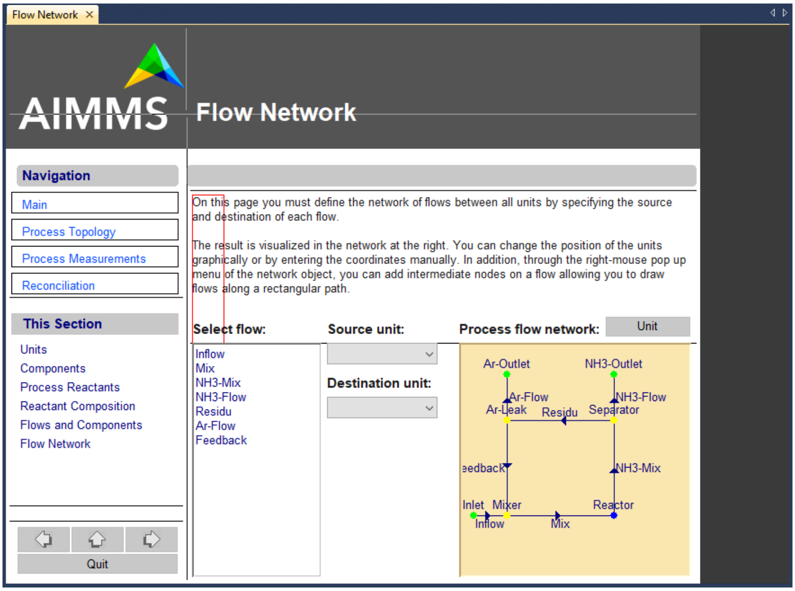
The result is visualized in the network at the right. You can change the position of the units graphically or by entering the coordinates manually. In addition, through the right-mouse pop up menu of the network object, you can add intermediate nodes on a flow allowing you to draw flows along a rectangular path.

Figure 13.3: End-user page associated with Figure 13.1

- The header area will have a fixed height at the top of the page whatever the page height, but will grow (or shrink) along with the page width.
- Similarly, the entire footer area will remain a fixed distance from the bottom of the page, and grow along with the page width.
- The information on the left-hand side of the data area has a fixed width, and the table will only grow/shrink vertically along with the page height.
- The flow chart header on the right-hand side of the data area has a fixed height, while the flow chart itself will grow/shrink along with both the page height and width.

Original size only

When entering **Edit** mode, AIMMS will always restore the editable page area to its original size (as saved at page creation time). This ensures that objects placed on the page always use the same coordinate system, preventing pixel rounding problems during a page resize. If the page has been saved at a different end-user size, AIMMS will open the page frame at the latest end- user size, and make the parts outside the original (editable) page size unavailable for editing, as illustrated in Figure 13.4. Any split line added to a page (or to its templates), will be visible in a page in **Edit** mode as a thin line.



On this page you must define the network of flows between all units by specifying the source and destination of each flow.

The result is visualized in the network at the right. You can change the position of the units graphically or by entering the coordinates manually. In addition, through the right-mouse pop up menu of the network object, you can add intermediate nodes on a flow allowing you to draw flows along a rectangular path.

Figure 13.4: Editable area and split lines of a resizable page in **Edit** mode

13.2 Resizable templates

Creating resizable templates

When you are creating an AIMMS-based application with many resizable pages, all based on a number of page templates, you should also consider defining the basic resize properties of these templates. As templates behave as ordinary pages in the template tree, you can add split lines to templates as described in the previous section.

Inherited resizability

All templates and end-user pages based upon a resizable template inherit the resize properties of that template, i.e. all split lines in the template are also applicable to its child templates and pages. Generally, such inherited split lines should take care of the resize properties of those objects that are contained in the template itself.

Adding split lines

On any page (either template or end-user page) you can always add additional split lines to those inherited from its ancestor template(s). The added split lines are used to specify the resize properties of the additional objects that have been placed on the page. In this manner, the template tree can be used to define the entire look-and-feel of your pages in a hierarchical manner, and their resize properties.

Example revisited

The example page in Figures 13.1 and 13.3 already illustrates the inherited resizability from templates. In fact, Figure 13.1 displays the split line configuration of a template defining the common header and footer area of all its child pages. The page in Figure 13.3, which uses this template, automatically inherits its resize properties. Therefore, the table in the "data area" of this page automatically grows or shrinks in relation to the page size as dictated by the template.

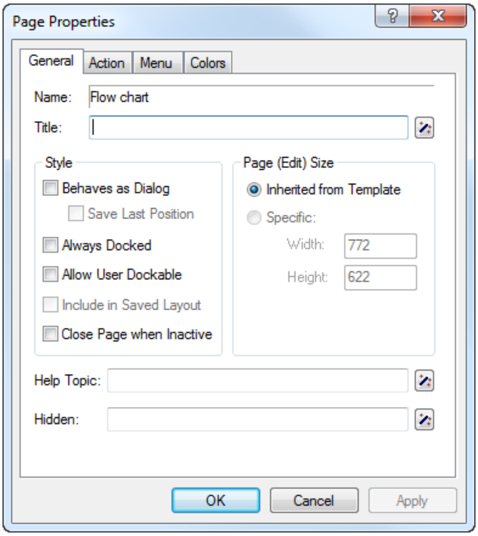
13.3 Adapting to changing screen resolutions

Coping with different resolutions

AIMMS allows you to create pages in such a manner that they will automatically adapt to changing screen resolutions. Thus, given a sensible configuration of split lines, you can create an application than can be run in resolutions other than the base resolution for which you developed the pages.

Page properties

To specify the behavior of pages and templates, open the **Properties** dialog box for the page (template), as illustrated in Figure 13.5. In the **Position & Size**



Name: Flow chart

Title:

Style

☐ Behaves as Dialog

☐ Save Last Position

☐ Always Docked

☐ Allow User Dockable

☐ Include in Saved Layout

☐ Close Page when Inactive

Page (Edit) Size

● Inherited from Template

○ Specific:

Width: 772

Height: 622

Help Topic:

Hidden:

Figure 13.5: The page **Properties** dialog box

area of this dialog box, you can select the default position and size of the page, which AIMMS will use when opening the page.

Opening modes

For every page in your application, you can select one of the four following standard page opening modes:

- get the position and size from the template used by the page,
- open the page at maximum size, but with visible page borders,
- open the page at maximum size, but without visible page borders, and
- open the page using the last saved position and size.

Inherited modes

If you specify that a page should obtain its position and page size from its template, the page will use the page open mode as specified for that template. When, in turn, this template has been specified to open according to its last saved position and size, an interesting interaction between the template and all its dependent pages will take place. Changing the position and size of *any* page using such a template will cause all the other pages using that template to be opened using the new position and size.

Resizable root template

As an application for the above, you could decide to make every page and page template dependent on the position and size of the root template. In this manner, changing the size of any page, will automatically result in the adjustment of every other page.

Save over sessions

When you have specified that a page or page template should save its last position, this position is stored between sessions. That is, the next time you open the same project, AIMMS will open such pages in the same positions as used in the previous sessions on the same computer.

Chapter 14

Creating Printed Reports

This chapter

Besides an attractive graphical end-user interface, paper reports containing the main model results are also an indispensable part of any successful modeling application. This chapter details printed reports. Printed reports are created and designed in a similar fashion to ordinary end-user pages, and can contain the same graphical objects for displaying data. There is, however, additional support for dividing large objects over multiple printed pages.

14.1 Print templates and pages

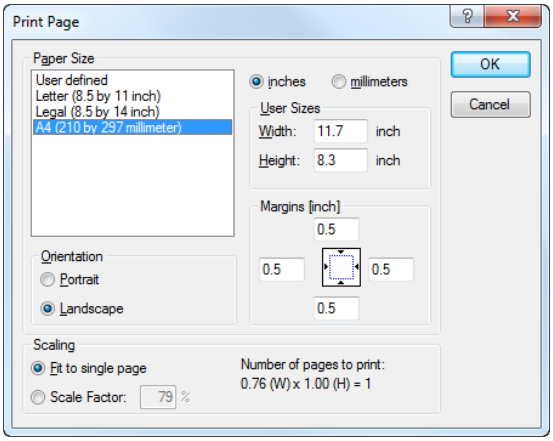
Printing versus GUI

AIMMS makes a distinction between *end-user* pages that are designed for interactive use by the end-user of your application and *print* pages that are specifically designed for printing on paper. While this may seem puzzling at first, a closer inspection reveals a number of serious drawbacks associated with printing ordinary end-user pages. The most important are:

- usually the screen resolution does not match the size of a sheet of paper,
- in a printed report, you cannot rely on the use of scroll bars on either the page itself or within objects if all the available information does not fit,
- the use of background colors may look nice on the screen, but often severely hinders the readability of printed reports, and
- you may want to add header and footer information or page numbers to printed pages, which are not part of an end-user page.

Printing ordinary end-user pages

Through the **File-Print** menu, AIMMS allows you to print a simple screen dump of the contents of any end-user page that currently is on the screen in your application. The **File-Print** menu will open the **Print Page** dialog box illustrated in Figure 14.1. Using this dialog box you can choose the size, border width and orientation of the screen dump to be produced. Any data that is not visible on the end-user page will also not appear in the screen dump.



Paper Size
- User defined
- Letter (8.5 by 11 inch)
- Legal (8.5 by 14 inch)
- A4 (210 by 297 millimeter)

● inches ○ millimeters

User Sizes
Width: 11.7 inch
Height: 8.3 inch

Margins [inch]
0.5
0.5 0.5
0.5

Orientation
○ Portrait
● Landscape

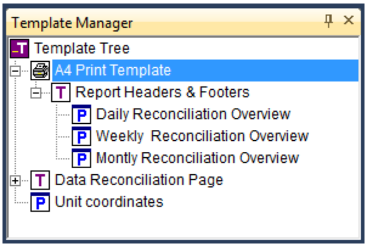
Scaling
● Fit to single page
○ Scale Factor: 79 %

Number of pages to print:
0.76 (W) x 1.00 (H) = 1

Figure 14.1: **Print Page** dialog box for end-user pages

Print templates and pages

An Aimms print page, on the other hand, allows you to compose a customized report with data objects that can automatically be resized to print all available object data. Print pages are characterized by the fact that they depend on a special *print template* in the template tree. You can add a print template via the **New-Print Template** item in the **Edit** menu of the **Template Manager**. Print templates can only be placed at the top level of the template tree, i.e. directly below the root, as illustrated in Figure 14.2. All pages below the print template behave as print pages.

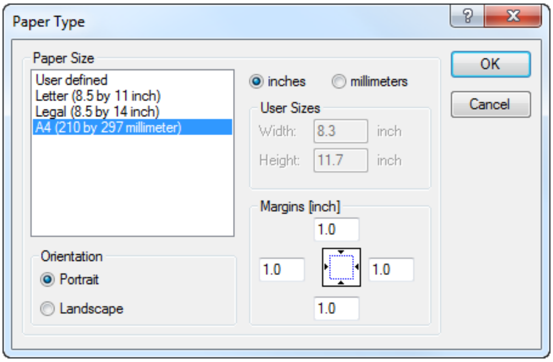


- Template Tree
  - A4 Print Template
    - Report Headers & Footers
      - Daily Reconciliation Overview
      - Weekly Reconciliation Overview
      - Montly Reconciliation Overview
  - Data Reconciliation Page
  - Unit coordinates

Figure 14.2: Example of a print template

Specifying paper type

Every print template has an associated paper type. The paper type lets you define properties such as paper size, paper orientation, and the width of the surrounding margins. By default, Aimms will create new print templates with the predefined A4 paper type. You can modify the paper type by opening the print template and selecting **Paper Type** in the **View** menu, which will then open the dialog box displayed in Figure 14.3. With it, you can either select one of the predefined paper types, or define a custom paper type by specifying the



Paper Type

Paper Size

- User defined
- Letter (8.5 by 11 inch)
- Legal (8.5 by 14 inch)
- A4 (210 by 297 millimeter)

● inches ○ millimeters

User Sizes

Width: 8.3 inch

Height: 11.7 inch

Margins [inch]: 1.0 1.0 1.0 1.0

Orientation

● Portrait

○ Landscape

Figure 14.3: **Paper Type** dialog box

paper size, orientation and margins yourself.

Page border

When you open a print page (or print template) in edit mode, AIMMS displays a rectangular box representing the margins corresponding to the current paper type. An example of an empty print page in landscape format containing a margin box is illustrated in Figure 14.4. The margin lines are not displayed when the page is previewed or printed. In edit mode, however, the margin lines may help you to position data objects within the printable area.

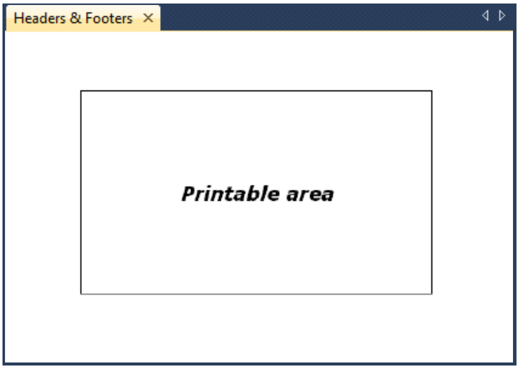


Figure 14.4: An empty print page in landscape format

Printing pages with margins

In general, AIMMS will print all objects on a print page, regardless of their placement with respect to the page margins. However, when you have indicated that a data object should be printed over multiple pages (as discussed in the next section), AIMMS will always restrict itself to printing within the indicated page margins.

Adding objects

You can add data objects and graphical objects to a print page in exactly the same way as you can add such objects to an ordinary end-user page. In fact, objects contained on your end-user pages which you want to be part of a printed report as well, can be copied directly to a print page by means of a simple copy and paste action.

Non-printable objects

You should note, however, that not all objects that are placed on a print page (and are visible on the screen) will be printed on paper. Specifically, AIMMS will omit all controls such as buttons and drop-down lists which are intended for interactive use only. Through such controls you can provide special facilities for your end-users such as allowing them to make last-minute choices prior to printing, activate the actual print job, or navigate to a previous or next print page through navigational controls linked to the page tree (see Section 12.1.2). To prevent interference with printable objects, non-printable controls are best placed in the page margins. Naturally, you can also place such controls on a separate dialog page.

Using additional templates

You can add one or more normal templates below any print template in the template tree, in exactly the same way as for ordinary end-user pages (see also Section 12.2). In this way, you can specify common components such as headers and footers, that are automatically inherited by all dependent print pages.

Displaying the page number

Page numbers can be added to a print page by displaying the predefined AIMMS identifier `CurrentPageNumber` either on the page itself or on any of its page templates. When printing a single page, AIMMS resets `CurrentPageNumber` to 1, and will number consecutively for any additional pages that are created because of a large data object. When printing a report that consists of multiple print pages (see below), AIMMS resets `CurrentPageNumber` to 1 prior to printing the report, and increments it for every printed page.

Printing print pages

AIMMS allows you to print a print page in several manners:

- when the page is opened on the screen, you can print it using the **File-Print** menu,
- you can attach the above action to a page or toolbar button, by adding the **File-Print** menu action to the button, or
- you can print the page from within the model using the `PrintPage` function.

Printing complete reports

In addition to printing single print pages, AIMMS also allows you to print entire reports consisting of multiple print pages. Printing such reports can be initiated only from within your model, through calls to the predefined functions `PrintStartReport`, `PrintPage` and `PrintEndReport`. A multipage report is started by a call to `PrintStartReport`, and finished by a call to `PrintEndReport`. All the single print pages constituting the report must be printed through consecutive calls to the `PrintPage` function in between. Such a composite report will be sent to the printer as a single print job, and by default all pages within the report will be numbered consecutively starting from 1. However, if you so desire, AIMMS allows you to modify the value of `CurrentPageNumber` between two consecutive calls to the `PrintPage` function. The print functions in AIMMS are discussed in more detail in Section 17.3.2.

14.2 Printing large objects over multiple pages

Printing large data objects

Print pages are explicitly designed to allow the printing of data objects that hold large amounts of data and, therefore, do not fit onto a single page. On a print page you can specify that such large objects should be split over as many pages as are needed to display all the underlying data, respecting the specified page margins. In addition, AIMMS allows you, in a flexible manner, to further restrict printing to those parts of the print page that are not already occupied by fixed page components such as headers and footers.

Required steps

In order to enable multipage printing, only two simple steps are required. More specifically, you should

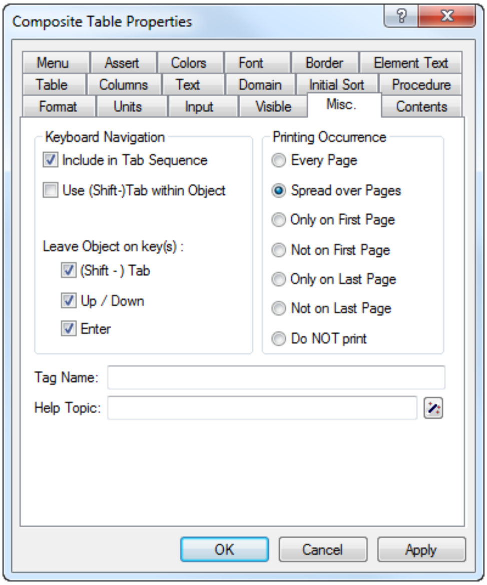
- modify the print properties of both the fixed page components and the large data objects contained on the page to specify their desired printing behavior, and
- create a simple subdivision of the print page by means of the resize split lines (see also Chapter 13) to specify how objects should be fixed to particular page positions, or resized as necessary.

The remainder of this section discusses both steps in more detail, and illustrates them on the basis of a realistic example.

Specify printing occurrence

For every object on a print page or template you can define when and how the object should be printed. Through the **Misc** tab of the object **Properties** dialog box (as displayed in Figure 14.5) you can specify that an object must be

- printed on every printed page (such as headers or footers),
- printed on only the first page or the last page,
- printed on all pages except for the first or the last page (ideal for indicating whether the report is continued or not),
- spread over as many pages as required to display all its associated data, or
- omitted from the printed output.



Keyboard Navigation

☑ Include in Tab Sequence

□ Use (Shift-)Tab within Object

Leave Object on key(s) :

☑ (Shift -) Tab

☑ Up / Down

☑ Enter

Printing Occurrence

○ Every Page

● Spread over Pages

○ Only on First Page

○ Not on First Page

○ Only on Last Page

○ Not on Last Page

○ Do NOT print

Tag Name:

Help Topic:

Figure 14.5: The **Misc** properties tab

Usage

Using these choices, you have the capability of having a single print page printed over multiple pages where each page component behaves as desired. For instance, headers and footers can be printed on every page or, perhaps, on all pages but the first. A report title needs only be displayed on the first page. Data objects which are expected to become large can be split over multiple pages.

Multiple page printing

By default, all objects will be printed the same size as they have been placed onto the print page during its creation. Thus, without further action, a large table is split over multiple pages based on the original table size. As you will see below, objects can be stretched to fill the entire print page by using AIMMS' resizability features.

Resizing page objects

Two types of split lines are useful when creating a resizable template for a printed report. *Fixed distance* split lines can be used to specify those areas of a page that contain components such as headers and footers which should keep their original shape and position. *Adjustable distance* split lines can be used to indicate that the objects contained in the relevant page area must be adapted to fill the maximum space within that area.

Spreading over multiple pages

Whenever a data object does not fit in an adjustable area, AIMMS will first extend the data object to the border of the adjustable area. This border may be either the page margin, or a fixed distance split line that has been placed on the page. When AIMMS runs into the border of an adjustable area, further printing of the data will continue on a new page. On the final page, AIMMS will reserve just enough space to contain the remaining data.

Multiple splits

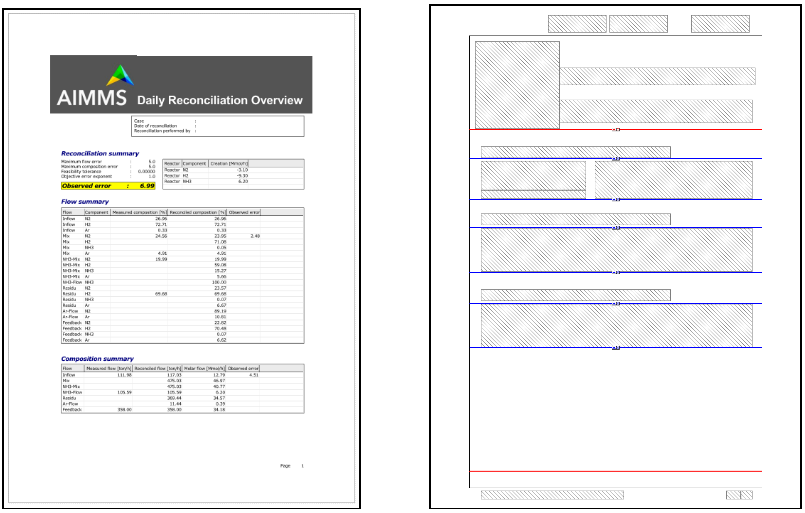
By creating multiple adjustable areas just below or alongside each other, you have the opportunity to place multiple data objects of varying size within a single report, with each object in its own resizable area. Once AIMMS has finished printing the object contained in the first resizable area, it will start printing the next object directly adjacent to the first, in either a left- to-right or top-to-bottom fashion, depending on your selected layout.

Use of templates

If you are creating multiple reports with more or less the same layout of headers and footers, you should preferably use template pages to define such fixed page components, together with their appropriate printing occurrence (e.g. first page only) and resizability properties for multiple page printing. If you use such templates wisely, creating a specific print page boils down to nothing more than adding one or more data objects to the data area of the page (i.e. the page area not occupied by a header and/or footer), and defining the appropriate print and resizability properties.

Example

The print page, and its corresponding configuration of split lines contained in Figure 14.6 illustrate AIMMS' capabilities of printing large data objects over multiple pages. In this page, the header and footer components are enclosed



AIMMS Daily Reconciliation Overview

Case :
Date of reconciliation :
Reconciliation performed by :

Reconciliation summary

Maximum flow error	:	5.0
Maximum composition error	:	5.0
Feasibility tolerance	:	0.00000
Objective error exponent	:	1.0
Observed error	**:**	**6.99**

Reactor	Component	Creation [Mmol/h]
Reactor	N2	-3.10
Reactor	H2	-9.30
Reactor	NH3	6.20

Flow summary

Flow	Component	Measured composition [%]	Reconciled composition [%]	Observed error
Inflow	N2	26.96	26.96	
Inflow	H2	72.71	72.71	
Inflow	Ar	0.33	0.33	
Mix	N2	24.56	23.95	2.48
Mix	H2		71.08	
Mix	NH3		0.05	
Mix	Ar	4.91	4.91	
NH3-Mix	N2	19.99	19.99	
NH3-Mix	H2		59.08	
NH3-Mix	NH3		15.27	
NH3-Mix	Ar		5.66	
NH3-Flow	NH3		100.00	
Residu	N2		23.57	
Residu	H2	69.68	69.68	
Residu	NH3		0.07	
Residu	Ar		6.67	
Ar-Flow	N2		89.19	
Ar-Flow	Ar		10.81	
Feedback	N2		22.82	
Feedback	H2		70.48	
Feedback	NH3		0.07	
Feedback	Ar		6.62	

Composition summary

Flow	Measured flow [ton/h]	Reconciled flow [ton/h]	Molar flow [Mmol/h]	Observed error
Inflow	111.98	117.03	12.79	4.51
Mix		475.03	46.97	
NH3-Mix		475.03	40.77	
NH3-Flow	105.59	105.59	6.20	
Residu		369.44	34.57	
Ar-Flow		11.44	0.39	
Feedback	358.00	358.00	34.18	

Page 1

Figure 14.6: Print page with in edit and resize edit mode

in areas which have a fixed size from the top and bottom margin, respectively, and are printed on every page of the report. The middle part of the page contains a number of data objects, each enclosed in an adjustable area from the top down. As a result, Aimms will split each object subsequently over as many pages as are necessary.

Chapter 15

Deploying End-User Applications

Deployment considerations

After a successful development phase of your AIMMS application, you have to start thinking about its deployment. For the application to become a successful end-user application too, you, as the application developer, need to consider issues like *protecting your intellectual property*, *authenticating your end-users*, and *distribution* of your application. AIMMS offers various tools to help you in all of these areas.

Application protection and authentication

AIMMS offers several levels of protection that can be applied to your application. To protect the content of your model from being viewed by the unauthorized users, AIMMS allows you to encrypt (parts of) your model. To protect the application against unauthorized access you can encrypt your application using the public keys of all authorized users. To further arrange the appropriate level of access within an organization, you can associate a user database with your AIMMS application, which can then be used to authenticate individual users and provide their level of access to your application. Protecting your AIMMS application and authenticating its use are discussed in full detail in Chapter 19.

Application distribution

To be able to run an AIMMS project, your users will need a copy of the project itself. To support easy project distribution, an AIMMS project can be compacted and distributed as a single-file project. In addition, your users need to have installed AIMMS on their computer. In order to enter input data and/or run the model, end-users need a commercial license of AIMMS.

AIMMS *PRO*

A more convenient way to distribute your application to your end-users is to use the AIMMS *P*ublishing and *R*emote *O*ptimization platform. AIMMS PRO makes it possible to deploy AIMMS applications to end-users quickly and efficiently through the AIMMS PRO Portal. More importantly, the AIMMS PRO Portal assures end-users can access the latest version of these AIMMS applications (and the corresponding versions of the AIMMS software) at all times through a web browser. More information on AIMMS PRO can be found in the AIMMS PRO User's Guide.

This chapter

This chapter gives some background on AIMMS end-user mode, and discusses `.aimmspack` project files, which you can use to distribute your AIMMS project as a single file.

15.1 Running end-user applications

Running end-user projects

An AIMMS project can run in two different modes, *developer* mode and *end-user* mode. While the developer mode allows you to use the full functionality described in this User's Guide, the end-user mode only allows you to *access* the end-user pages of the AIMMS project that were created in developer mode.

Disabled functionality

The AIMMS end-user mode lacks the essential tools for creating and modifying model-based applications. More specifically, the following tools are not available in end-user mode:

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

Thus, in end-user mode, there is no way in which an end-user can modify the contents of your AIMMS-based application.

Allowed usage

AIMMS end-users can only perform tasks specified by you as an application developer. Such tasks must be performed through data objects, buttons and the standard, or custom, end-user menus associated with the end-user pages in your project. They include:

- modifying the input data for your model in the end-user interface,
- executing procedures within your model to read data from an external data source, or performing a computation or optimization step,
- viewing model results in the end-user interface,
- writing model results to external data sources or in the form of printed reports, and
- performing case management tasks within the given framework of case types.

Thus, an end-user of your application does not need to acquire any AIMMS-specific knowledge. The only requirement is that the interface that you have created around your application is sufficiently intuitive and clear.

Requirements

Before you can distribute your AIMMS project as an end-user application, two requirements have to be fulfilled:

- you must have exported your application as an *end-user project* (see Section 19.1), and
- you need to associate a *startup page* with your application which will be displayed when your application is started by an end-user.

Assigning a startup page

For every end-user project, you must associate a single page within the project so that it becomes the project's *startup page*. Such an association can either be made directly by selecting a page for the 'Startup Page' option in the AIMMS **Options** dialog box (see Section 20.1), or implicitly as the first opened page in the *startup procedure* of the project using a call to the `PageOpen` function.

Role of startup page

After opening your project in end-user mode, AIMMS will display the startup page. As all communication between the end-user and your model is conducted through end-user pages of your design, this first page and/or its menus must provide access to all the other parts of your AIMMS application that are relevant for your end-users. If all pages are closed during a session, the end-user can still re-open the startup page using the first page button 1 on the **Project** toolbar, or via the **View-First Page** menu.

Startup procedure

In addition to a startup page you can also provide a startup procedure in the project-related AIMMS options. Inside the startup procedure you can perform any initializations necessary for an end-user to start working with the project. Such initializations can include setting up date or user related aspects of the project, or reading the data for particular identifiers from a database.

Replacing the splash screen

By default, AIMMS will display a splash screen during startup. When you are opening AIMMS with a particular project, you can replace AIMMS' own splash screen with a bitmap of your choice. If the project directory contains a bitmap (`.bmp`) file with the same name as the project file, AIMMS will display this bitmap file on the splash screen. In such a bitmap you can display, for instance, version information about your project.

15.2 Preparing an AIMMS application for distribution

`.aimmspack` project files . . .

A complete AIMMS application consists of several files (see Section 2.5 for an overview of these 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 a single-file project, by packing all relevant files into a single file with the `.aimmspack` extension.

Creating a .aimmspack file

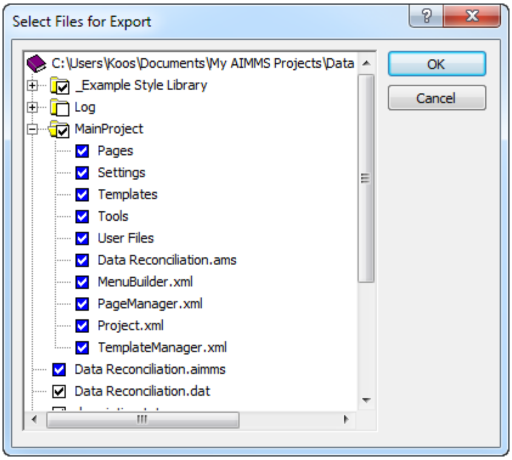
Turning your project into a single-file, end-user project is very straightforward. Just select the **File-Export End User Project** menu command which will open the **Select destination .aimmspack File** dialog box which allows you to specify the name and location of the file to be exported.

Encryption

After having specified a destination file, the **Encryption of Exported End-User Project** dialog box opens which allows you the security setting for your end-user project. Encryption is described in more detail in Section 19.1.

The .aimmspack file contents

By default AIMMS will select all files in the project directory to be included in the `.aimmspack` file while ignoring all files in the `Backup` and `Log` directories. All files associated with referenced library projects will also be included. The project (`.aimms`) and all source files (e.g. `.ams` and `.xml` files) for the main project and all of the library projects involved are mandatory while the settings for all other files can be changed through the **Select Files for Export** dialog box (see Figure 15.1. Note that only include files (and folders) that are contained in the main project folder and folders of library projects. If your project refers to files in other locations you must make sure that these files exist on the destination computer.



Select Files for Export

C:\Users\Koos\Documents\My AIMMS Projects\Data

- ☑ _Example Style Library
- □ Log
- ☑ MainProject
  - ☑ Pages
  - ☑ Settings
  - ☑ Templates
  - ☑ Tools
  - ☑ User Files
  - ☑ Data Reconciliation.ams
  - ☑ MenuBuilder.xml
  - ☑ PageManager.xml
  - ☑ Project.xml
  - ☑ TemplateManager.xml
- ☑ Data Reconciliation.aimms
- ☑ Data Reconciliation.dat

Figure 15.1: The **Select Files for Export** dialog box

Running an .aimmspack project file

The `.aimmspack` project files are dealt with as if they were ordinary AIMMS project files. Both the developer and end-user version of AIMMS are capable of running both `.aimms` and `.aimmspack` files.

Unpacking an `.aimmspack` *project file ...*

When a *developer* opens an `.aimmspack` project file, he will *always* be prompted for a location where to unpack to. On the other hand, when an *end-user* opens an `.aimmspack` project file, only the *first time*, he will be prompted for a location where to unpack to. Any subsequent time an end-user opens the `.aimmspack` file, AIMMS will look whether the location where the `.aimmspack` file was unpacked previously contains the unpacked version of the same `.aimmspack` file. If so, AIMMS will open the unpacked version without user interaction. Otherwise, AIMMS will prompt the end-user for a (new) location, unpack the `.aimmspack` project and run it with AIMMS. So, when you send your end-user an updated version of a packed project, AIMMS will notice the version change and prompt the end-user with a question whether or not to overwrite the existing project with the new version.

... using the command line

Alternatively, you can indicate where you want the `.aimmspack` file to be unpacked via the commandline argument *--unpack-folder*.

Selecting an AIMMS *version*

If multiple AIMMS versions have been installed on the computer, you can select the particular AIMMS version to use on an `.aimms` or `.aimmspack` file through the right-mouse popup menu in Windows Explorer. If you double-click an `.aimms` or `.aimmspack` the `AimmsLauncher` utility program will try to open the AIMMS version that most closely matches the AIMMS version that was used to create the project.

Part IV

Data Management

Chapter 16

Case Management

This chapter

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:

- Chapter 27 ”Communicating With Databases”
- Chapter 29 ”Reading and Writing Spreadsheet Data”
- Chapter 30 ”Reading and Writing XML Data”
- Chapter 34 ”The Aimms Programming Interface”

Furthermore, the AIMMS SDK offers access to AIMMS data from Java, C#, and C++, see `http://download.aimms.com/aimms/AimmsSDK`.

16.1 Working with cases

Case management tasks

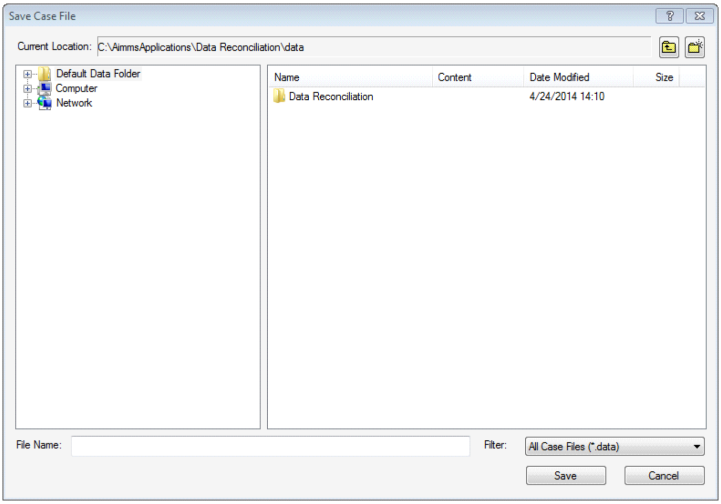
A case file is a single file containing the data of some identifiers in an AIMMS model. The **Data** menu is the main tool through which you can accomplish tasks such as saving, loading, merging, and comparing case files. This menu item is part of the developer menu and is available by default on all 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 Figure 16.1. 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.



Current Location: C:\AimmsApplications\Data Reconciliation\data

Name	Content	Date Modified	Size
Data Reconciliation		4/24/2014 14:10	

File Name:

Filter: All Case Files (*.data)

Figure 16.1: The **Save Case File** dialog box

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.

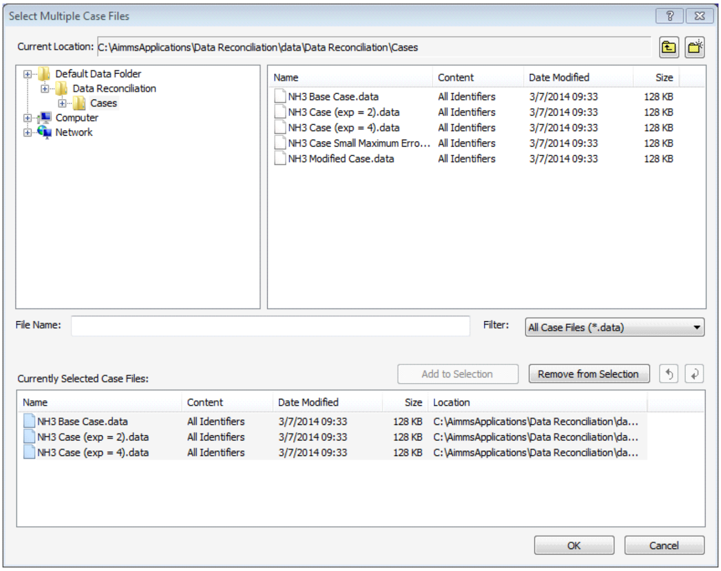
16.2 Managing multiple case selections

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 Figure 16.2. 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.



Select Multiple Case Files
?
Current Location: C:\AimmsApplications\Data Reconciliation\data\Data Reconciliation\Cases
Default Data Folder
Data Reconciliation
Cases
Computer
Network
File Name:
Filter:
All Case Files (*.data)
Currently Selected Case Files:
Add to Selection
Remove from Selection
OK
Cancel

Name	Content	Date Modified	Size
NH3 Base Case.data	All Identifiers	3/7/2014 09:33	128 KB
NH3 Case (exp = 2).data	All Identifiers	3/7/2014 09:33	128 KB
NH3 Case (exp = 4).data	All Identifiers	3/7/2014 09:33	128 KB
NH3 Case Small Maximum Erro...	All Identifiers	3/7/2014 09:33	128 KB
NH3 Modified Case.data	All Identifiers	3/7/2014 09:33	128 KB

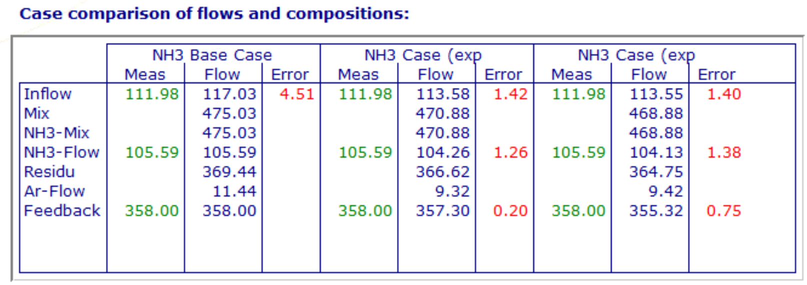
Name	Content	Date Modified	Size	Location
NH3 Base Case.data	All Identifiers	3/7/2014 09:33	128 KB	C:\AimmsApplications\Data Reconciliation\da...
NH3 Case (exp = 2).data	All Identifiers	3/7/2014 09:33	128 KB	C:\AimmsApplications\Data Reconciliation\da...
NH3 Case (exp = 4).data	All Identifiers	3/7/2014 09:33	128 KB	C:\AimmsApplications\Data Reconciliation\da...

Figure 16.2: The **Select Multiple Case Files** dialog box

16.2.1 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. Figure 16.3 is an illustration of a table which displays the contents of a single identifier for all the cases in the case selection shown in Figure 16.2.



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

Figure 16.3: 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. Figure 16.4 illustrates the object-specific **Properties** dialog box of a table object. 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.

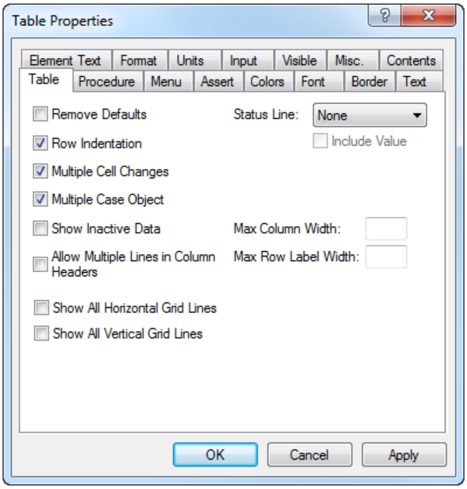


Table Properties

☐ Remove Defaults
☑ Row Indentation
☑ Multiple Cell Changes
☑ Multiple Case Object
☐ Show Inactive Data
☐ Allow Multiple Lines in Column Headers
☐ Show All Horizontal Grid Lines
☐ Show All Vertical Grid Lines

Status Line: None
☐ Include Value
Max Column Width:
Max Row Label Width:

Figure 16.4: Table-specific **Properties** dialog box

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.

16.2.2 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 Section 17.3.5, 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 AllCases 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 Section 25.3, at the start of procedures containing case referencing.

16.3 Working with selections of identifiers

Case content type

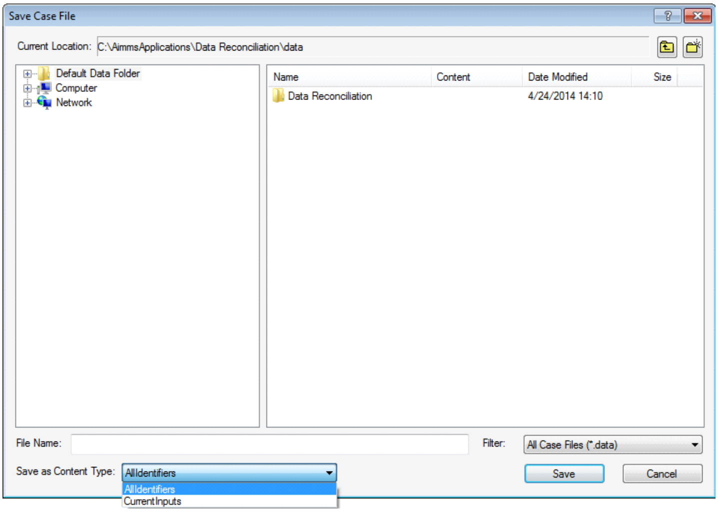
Next to saving the contents *all* identifiers in a case file, it is also possible to save the data of *a selection of* identifiers in a case file. Such a selection of identifiers to be saved to a case file is called a *case content type*. A case content type is a subset of AllIdentifiers.

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 Figure 16.5 where the **Save Case File** dialog box allows you to select between the case content types AllIdentifiers and CurrentInputs.



Save Case File

Current Location: C:\AimmsApplications\Data Reconciliation\data

Name	Content	Date Modified	Size
Data Reconciliation		4/24/2014 14:10	

File Name:

Filter: All Case Files (*.data)

Save as Content Type: AllIdentifiers

Figure 16.5: The **Save Case File** dialog box offering content types

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.

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.

Part V

Miscellaneous

Chapter 17

User Interface Language Components

This chapter

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

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

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

17.1 Updatability of identifiers

Dynamic control required

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

Multiple phases in your application

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

Indicating input and output status

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

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

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

The set CurrentInputs

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

The set AllUpdatable-Identifiers

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

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

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

17.2 Setting colors within the model

Color as indicator

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

Setting colors in the model

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

The set AllColors

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

Color parameters

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

Example

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

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

17.2.1 Creating non-persistent user colors

Non-persistent user colors

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

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

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

Adding non-persistent colors

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

Deleting and modifying colors

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

Retrieving RGB-values

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

17.3 Interfacing with the user interface

Interface functions

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

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

Return values

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

Limited use in certain cases

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

Obtaining the error message

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

Example

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

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

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

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

17.3.1 Page functions

Model page control

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

Page functions

The following functions for manipulating pages are available in AIMMS.

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

17.3.2 Print functions

Printing facilities

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

Print functions

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

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

17.3.3 File functions

File manipulation

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

File functions

The following functions are available for file manipulation in AIMMS.

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

Directory functions

The following functions are available for directory manipulation.

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

17.3.4 Dialog box functions

Two types of dialog boxes

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

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

In addition to these standard dialog boxes available in AIMMS, it is also possible to create customized dialog boxes using dialog pages (see Section 11.3), and open these using the `PageOpen` function discussed in Section 17.3.1.

Information dialog boxes

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

- `DialogMessage`(*message*[,*title*]), and `DialogError`(*message*[,*title*]) Both show *message* until **OK** button is pressed. They differ in icons displayed.

- `DialogAsk`(*message*,*button1*,*button2*[,*button3*]) Show *message* and offer two or three choices.
- `DialogProgress`(*message*[,*percentage*]) Show *message* and progress bar. Execution is continued.
- `StatusMessage`(*message*) Show *message* at the bottom of the AIMMS window.

Data entry dialog boxes

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

- `DialogGetString`(*message*,*reference*[,*title*]) Get a string.
- `DialogGetElement`(*title*,*reference*)
- `DialogGetElementByText`(*title*,*reference*,*element-text*)
- `DialogGetElementByData`(*title*,*reference*,*element-data*)
- `DialogGetNumber`(*message*,*reference*[,*decimals*][,*title*])
- `DialogGetPassword`(*message*,*reference*[,*title*])
- `DialogGetDate`(*title*,*date-format*,*date*[,*nr-rows*][,*nr-columns*])

17.3.5 Case management functions

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

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

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

Basic case functions

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

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

- CaseCreateDifferenceFile(*case*,*filename*,*diff-types* ,*absolute-tolerance*,*relative-tolerance*,*output-precision*)

Here the arguments are:

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

Case dialog functions

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

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

Here the arguments are:

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

Data manamement functions

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

Data menu functions

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

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

- CaseCommandNew() The default action behind the **Data - New Case** menu item.
- CaseCommandSave() The default action behind the **Data - Save Case** menu item.
- CaseCommandSaveAs() The default action behind the **Data - Save Case As** menu item.

Case file related identifiers

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

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

17.3.6 Execution control functions

Execution control

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

Control functions

The following execution control functions are available in AIMMS.

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

17.3.7 Debugging information functions

Debugging information

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

Execution information functions

The following execution information functions are available in Aimms.

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

Profiler control

The following profiler control functions are available in Aimms.

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

17.3.8 Obtaining license information

License information functions

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

License functions

The following licensing functions are available in Aimms.

- `LicenseNumber`(*license*)
- `LicenseStartDate`(*date*)
- `LicenseExpirationDate`(*date*)
- `LicenseMaintenanceExpirationDate`(*date*)
- `LicenseType`(*type*,*size*)
- `AimmsRevisionString`(*revision*)

Chapter 18

Calling AIMMS

This chapter

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.

18.1 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 18.1 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.

Specifying a user

When an AIMMS project is linked to an end-user database (see Chapter 19), 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.

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
--max-threads		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

Table 18.1: AIMMS command line options

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 Section 20.1).

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 (see also Section 2.6.4). 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 (see also Section 2.6). 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 Section 19.2 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 Chapter 34 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 (see alse Section 15.2). 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 `.aimmspack` 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 `.aimmspack` file, AIMMS will ask for the folder where you want the `.aimmspack` 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 (see Section 15.1), 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 Section 17.3.6.

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.

18.2 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:

- when linking from within an Excel spreadsheet, use the Excel Add-In (see The Excel Add-In User's Guide), or
- link directly against the AIMMS API (see Chapter 34 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.

18.3 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 ;
```

Chapter 19

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?

This chapter

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.

19.1 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 `.aimmspack` file (see also Section 15.2). 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 `.aimmspack` 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 `.aimmspack` file.

. . . and encrypt

Having specified the name for the `.aimmspack` file, the **Encryption of Exported End-User Project** dialog box (as illustrated in Figure 19.1) opens and allows you to add encryption to the exported version of your application. 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.

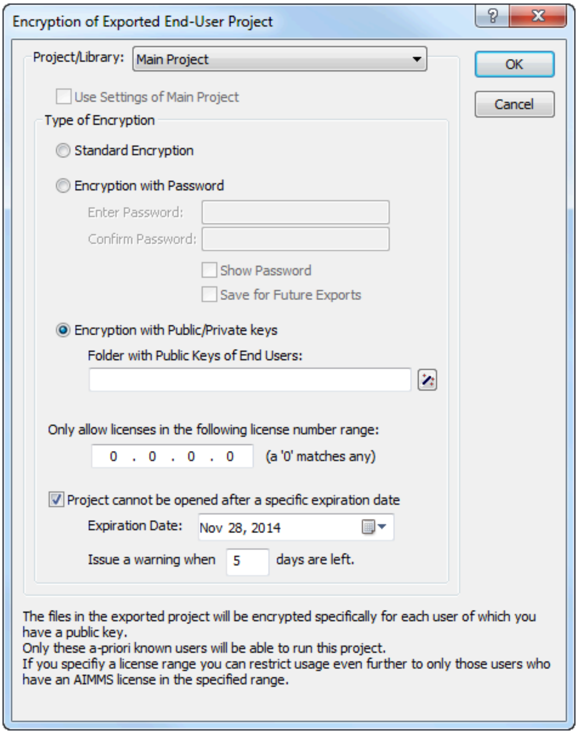
Add an expiration date

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.

19.1.1 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.



Encryption of Exported End-User Project

Project/Library: Main Project

□ Use Settings of Main Project

Type of Encryption

○ Standard Encryption

○ Encryption with Password

Enter Password:

Confirm Password:

□ Show Password

□ Save for Future Exports

● Encryption with Public/Private keys

Folder with Public Keys of End Users:

Only allow licenses in the following license number range:

0 . 0 . 0 . 0 (a '0' matches any)

☑ Project cannot be opened after a specific expiration date

Expiration Date: Nov 28, 2014

Issue a warning when 5 days are left.

The files in the exported project will be encrypted specifically for each user of which you have a public key.
Only these a-priori known users will be able to run this project.
If you specify a license range you can restrict usage even further to only those users who have an AIMMS license in the specified range.

Figure 19.1: The **Encryption of Exported End-User Project** dialog box

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.

19.1.2 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.

19.2 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 User's Guide `http://manual.aimms.com/pro/` 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.

Chapter 20

Project Settings and Options

This chapter

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.

20.1 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.

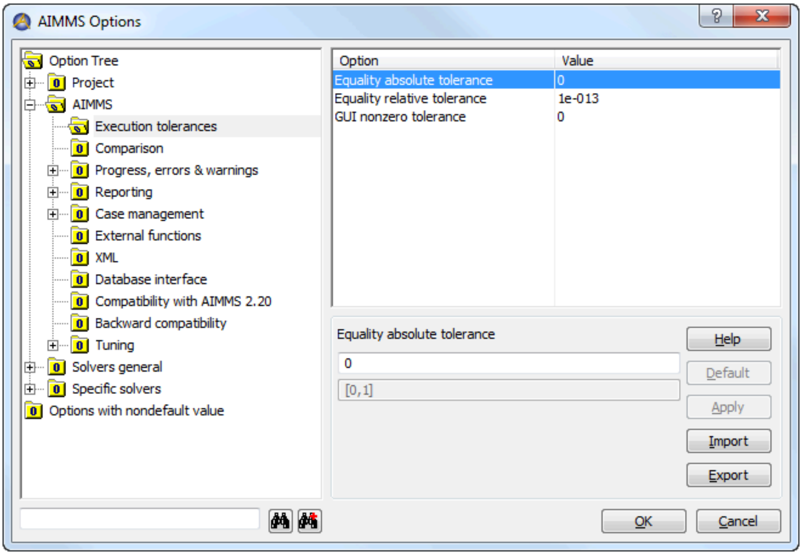
Option types

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 Figure 20.1. In this dialog box, an *option tree* lists all available AIMMS execution and solver options in a hierarchical fashion.



Option	Value
Equality absolute tolerance	0
Equality relative tolerance	1e-013
GUI nonzero tolerance	0

Equality absolute tolerance

0

[0,1]

Figure 20.1: The AIMMS **Options** dialog box

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 Figure 20.1, 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

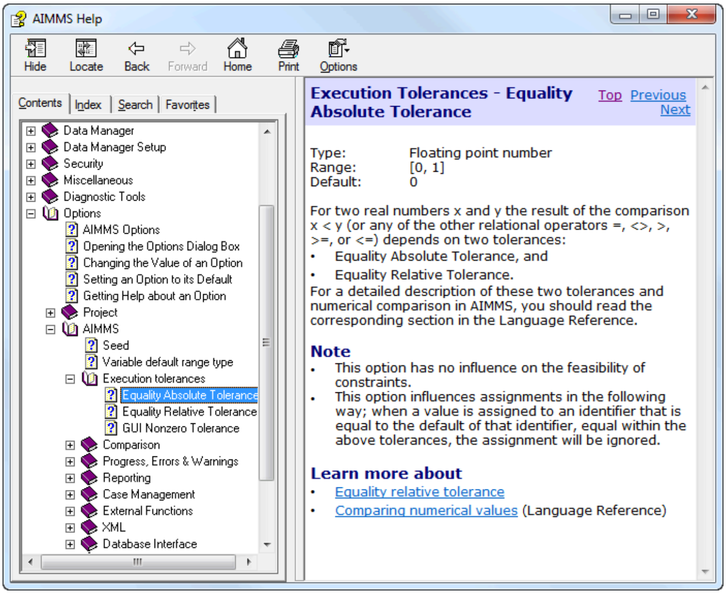
- a simple edit field,
- radio buttons,
- a drop-down list, or
- a wizard in the case where the value of an option is model-related.

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 Figure 20.2, this will open a help window containing a more detailed description of the selected option.



Execution Tolerances - Equality Absolute Tolerance

Type: Floating point number
Range: [0, 1]
Default: 0

For two real numbers x and y the result of the comparison x < y (or any of the other relational operators =, <>, >, >=, or <=) depends on two tolerances:

- Equality Absolute Tolerance, and
- Equality Relative Tolerance.

For a detailed description of these two tolerances and numerical comparison in AIMMS, you should read the corresponding section in the Language Reference.

Note

- This option has no influence on the feasibility of constraints.
- This option influences assignments in the following way; when a value is assigned to an identifier that is equal to the default of that identifier, equal within the above tolerances, the assignment will be ignored.

Learn more about

- Equality relative tolerance
- Comparing numerical values (Language Reference)

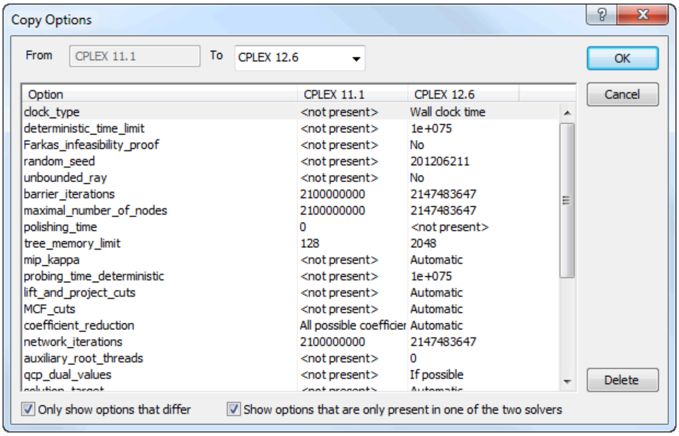
Figure 20.2: Option help

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 Section 20.3 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 Figure 20.3. 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.



From CPLEX 11.1 To CPLEX 12.6

Option	CPLEX 11.1	CPLEX 12.6
clock_type	<not present>	Wall clock time
deterministic_time_limit	<not present>	1e+075
Farkas_infeasibility_proof	<not present>	No
random_seed	<not present>	201206211
unbounded_ray	<not present>	No
barrier_iterations	2100000000	2147483647
maximal_number_of_nodes	2100000000	2147483647
polishing_time	0	<not present>
tree_memory_limit	128	2048
mip_kappa	<not present>	Automatic
probing_time_deterministic	<not present>	1e+075
lift_and_project_cuts	<not present>	Automatic
MCF_cuts	<not present>	Automatic
coefficient_reduction	All possible coefficier	Automatic
network_iterations	2100000000	2147483647
auxiliary_root_threads	<not present>	0
qcp_dual_values	<not present>	If possible
solution_target	<not present>	Automatic

☑ Only show options that differ ☑ Show options that are only present in one of the two solvers

Figure 20.3: 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 AIMMS Language Reference. 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.

20.2 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.

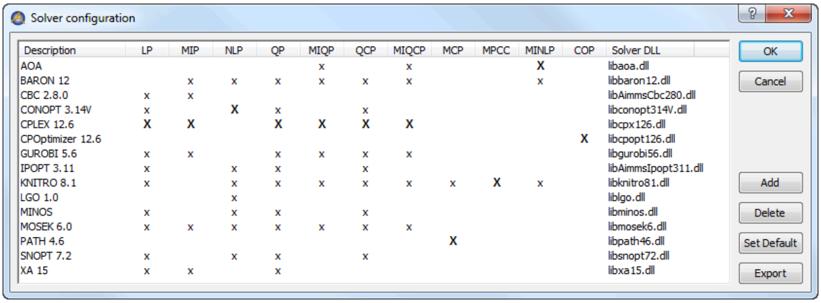
20.3 Solver configuration

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 Figure 20.4. 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.



Description	LP	MIP	NLP	QP	MIQP	QCP	MIQCP	MCP	MPCC	MINLP	COP	Solver DLL
AOA					x		x			X		libaoa.dll
BARON 12		x	x	x	x	x	x			x		libbaron12.dll
CBC 2.8.0	x	x										libAimmsCbc280.dll
CONOPT 3.14V	x		X	x		x						libconopt314V.dll
CPLEX 12.6	X	X		X	X	X	X					libcpx126.dll
CPOptimizer 12.6											X	libcpopt126.dll
GUROBI 5.6	x	x		x	x	x	x					libgurobi56.dll
IPOPT 3.11	x		x	x		x						libAimmsIpopt311.dll
KNITRO 8.1	x		x	x	x	x	x	x	X	x		libknitro81.dll
LGO 1.0			x									liblgo.dll
MINOS	x		x	x		x						libminos.dll
MOSEK 6.0	x	x	x	x	x	x	x					libmosek6.dll
PATH 4.6								X				libpath46.dll
SNOPT 7.2	x		x	x		x						libsnopt72.dll
XA 15	x	x		x								libxa15.dll

Figure 20.4: 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:

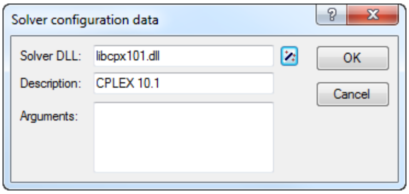
- modify the default solver for a particular model type, and
- add or delete solvers.

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 Figure 20.5. 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.


Solver configuration data

Solver DLL: libcpx101.dll

Description: CPLEX 10.1

Arguments:

Figure 20.5: The **Solver Configuration Data** dialog box

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.

20.4 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 (see Chapter 14), 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

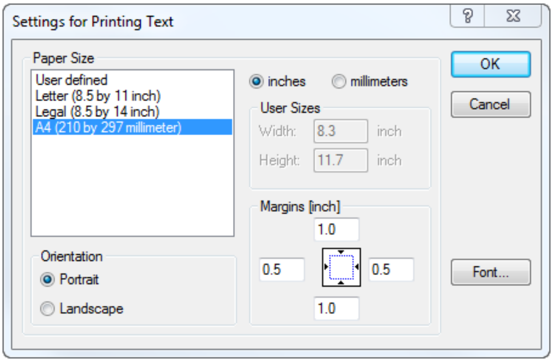
End-user pages and print pages are printed according to the settings that you have selected for these pages. These settings include:

- the selection of the paper type on which pages are printed (see Section 14.1), and
- the selection of object fonts and colors through the AIMMS font and color selection dialog boxes (see Section 11.2).

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 Figure 20.6.



Settings for Printing Text

Paper Size

- User defined
- Letter (8.5 by 11 inch)
- Legal (8.5 by 14 inch)
- A4 (210 by 297 millimeter)

● inches ○ millimeters

User Sizes

Width: 8.3 inch

Height: 11.7 inch

Margins [inch]: 1.0, 0.5, 0.5, 1.0

Orientation

● Portrait

○ Landscape

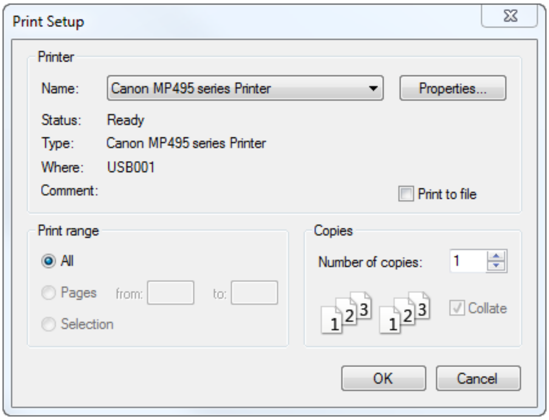
Figure 20.6: The **Text Printing** dialog box

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 Figure 20.7.



Print Setup

Printer

Name: Canon MP495 series Printer

Status: Ready

Type: Canon MP495 series Printer

Where: USB001

Comment:

□ Print to file

Print range

● All

○ Pages from: to:

○ Selection

Copies

Number of copies: 1

☑ Collate

Figure 20.7: 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.

Chapter 21

Localization 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

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.

21.1 Localization of end-user interfaces

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

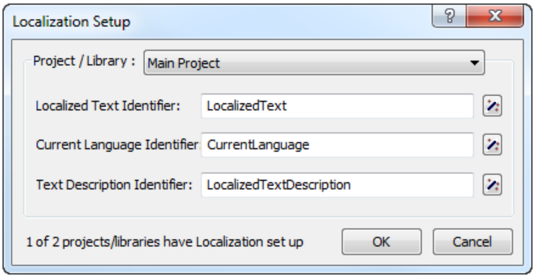
If your application consist of multiple library projects (see also Chapter 3), developed and maintained by different modelers, each of these libraries can have its own **Localization** section and identifiers to store its localization strings. When performing the localization conversion on a library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

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 Figure 21.1, 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.



Project / Library : Main Project
Localized Text Identifier: LocalizedText
Current Language Identifier CurrentLanguage
Text Description Identifier: LocalizedTextDescription
1 of 2 projects/libraries have Localization set up

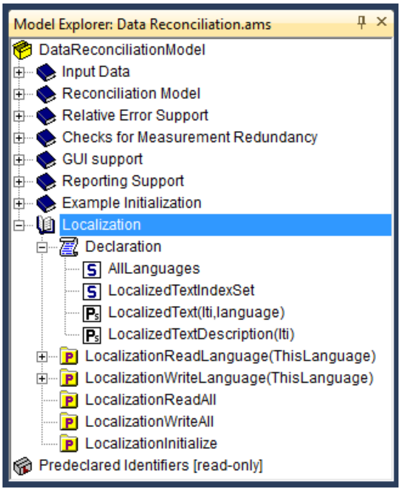
Figure 21.1: Setting up localization support

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 Figure 21.2. The declaration section contained in it declares the default set and string parameters used for storing all localization information.



Model Explorer: Data Reconciliation.ams

DataReconciliationModel
- Input Data
- Reconciliation Model
- Relative Error Support
- Checks for Measurement Redundancy
- GUI support
- Reporting Support
- Example Initialization
- Localization
  - Declaration
    - AllLanguages
    - LocalizedTextIndexSet
    - LocalizedText(lti,language)
    - LocalizedTextDescription(lti)
  - LocalizationReadLanguage(ThisLanguage)
  - LocalizationWriteLanguage(ThisLanguage)
  - LocalizationReadAll
  - LocalizationWriteAll
  - LocalizationInitialize

Predeclared Identifiers [read-only]

Figure 21.2: Localization section in the model tree

- 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.

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

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

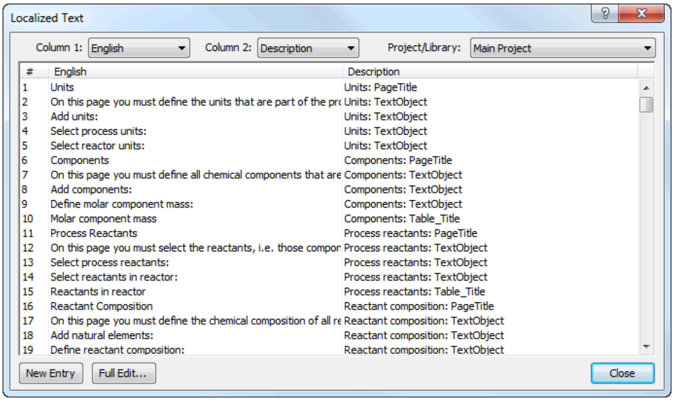
During the localization conversion, AIMMS will warn for any duplicate string it encounters. For such duplicate strings, you have the opportunity to create a new entry in the **Localized Text Identifier** or to re-use an existing entry. Re-using existing entries can be convenient for common strings such 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 Figure 21.3. In this dialog box, AIMMS displays a numbered list of all localized strings, along with the description of the origin of each string. The string numbers exactly correspond to the elements of the set `LocalizedTextIndexSet` discussed above.

Modifying dialog box contents

Through the drop down lists at the top of the **Localized Text** dialog box of Figure 21.3, you can select the contents of the first and second string columns, respectively. For each column, you can select whether to display the localized text for any language defined in the set `AllLanguages`, or the description associated with each string. By viewing the localized strings for two languages alongside, you can easily provide the translation of all localized strings for a



Column 1: English

Column 2: Description

Project/Library: Main Project

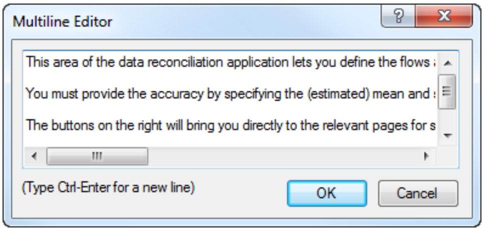
#	English	Description
1	Units	Units: PageTitle
2	On this page you must define the units that are part of the pr	Units: TextObject
3	Add units:	Units: TextObject
4	Select process units:	Units: TextObject
5	Select reactor units:	Units: TextObject
6	Components	Components: PageTitle
7	On this page you must define all chemical components that are	Components: TextObject
8	Add components:	Components: TextObject
9	Define molar component mass:	Components: TextObject
10	Molar component mass	Components: Table_Title
11	Process Reactants	Process reactants: PageTitle
12	On this page you must select the reactants, i.e. those compon	Process reactants: TextObject
13	Select process reactants:	Process reactants: TextObject
14	Select reactants in reactor:	Process reactants: TextObject
15	Reactants in reactor	Process reactants: Table_Title
16	Reactant Composition	Reactant composition: PageTitle
17	On this page you must define the chemical composition of all re	Reactant composition: TextObject
18	Add natural elements:	Reactant composition: TextObject
19	Define reactant composition:	Reactant composition: TextObject

Figure 21.3: The **Localized Text** dialog box

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 Figure 21.4. 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.



This area of the data reconciliation application lets you define the flows a

You must provide the accuracy by specifying the (estimated) mean and s

The buttons on the right will bring you directly to the relevant pages for s

(Type Ctrl-Enter for a new line)

Figure 21.4: The **Multineline Editor** 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 Figure 21.5 Invoking this wizard will open the **Localized Text** dialog box illustrated in Figure 21.3, in which you can either select an existing localized string, or create a new entry through the **New Entry** button. Notice that the **Localized Text** wizard only shows the localization strings for the main or library project you are currently editing, and any of the included library projects which have the localization identifiers in their public interface. After closing the dialog box, AIMMS will add a reference to the localized text identifier in the edit field of the property for which you invoked the wizard, corresponding to the particular string selected in the **Localized Text** dialog box.

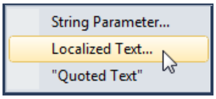


Figure 21.5: The **Localized Text** wizard

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 Section 2.5.1). The localized strings for every language, as well as the string descriptions are stored in separate user project files, as illustrated in Figure 21.6. The read and write statements in the bodies of the localization procedures added to the **Localization** section of your model, assume this structure of project user files for localization support.

Automatically updated

Whenever you use the **Localized Text** dialog box of Figure 21.3, either through the **Tools-Localization-Show Strings** menu or by invoking the **Localized Text** wizard, AIMMS will make sure that the contents of appropriate localization data files are read in before displaying the localization data for a particular language. Likewise, AIMMS will make sure that the contents of the appropriate project user files are updated when you close the **Localized Text** dialog box.

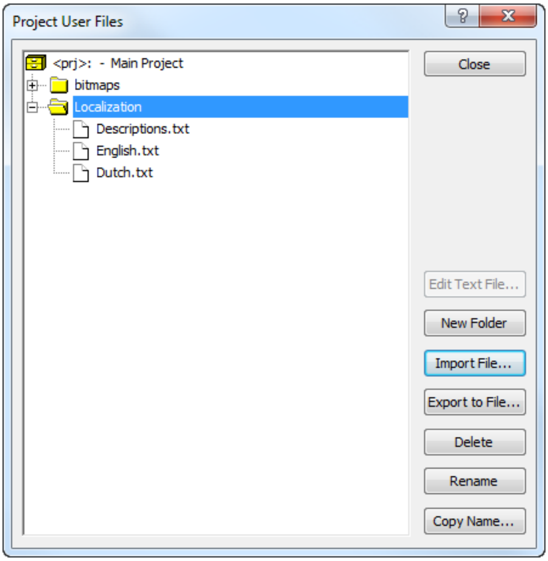


Figure 21.6: Default of localization data as user project files

Manual edits

By using the import and export facilities for project user files (see also Section 2.5.1), 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.

Appendices

Index