AIMMS User's Guide - Model Explorer

Aimms 4

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

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

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.

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

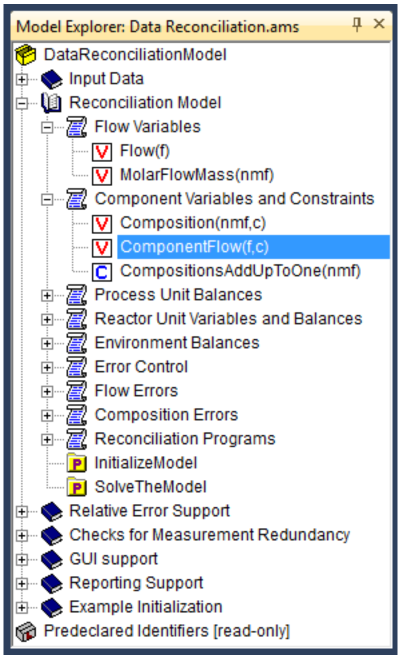
Decision making commonly requires access to massive amounts of information on which to base the decision making process. As a result, professional decision support systems are usually very complex programs with hundreds of (indexed) identifiers to store all the data that are relevant to the decision 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
      - V Flow(f)
      - V MolarFlowMass(nmf)
    - Component Variables and Constraints
      - V Composition(nmf,c)
      - V ComponentFlow(f,c)
      - C CompositionsAddUpToOne(nmf)
    - Process Unit Balances
    - Reactor Unit Variables and Balances
    - Environment Balances
    - Error Control
    - Flow Errors
    - Composition Errors
    - Reconciliation Programs
    - P InitializeModel
    - P 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

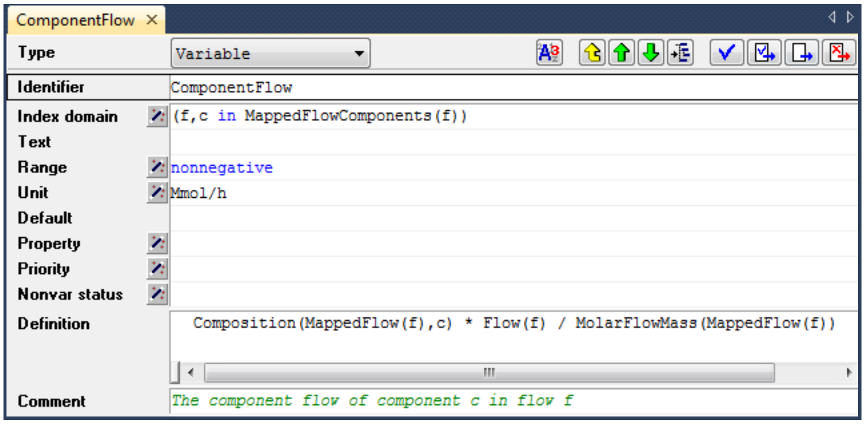
Another frequently occurring node type is the declaration of a procedure or a function. Such a procedure or function node contains the data retrieval statements, computations, and algorithms that make up the procedural execution of your modeling application. Procedures and functions are represented by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



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

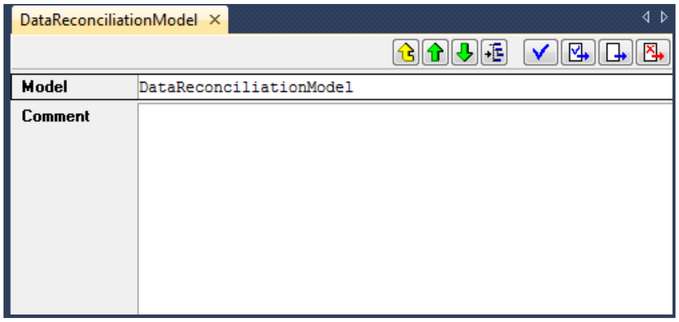
When you feel that particular groups of declarations, procedures and functions belong together in a logical manner, you are encouraged to create a new structuring section with a descriptive name within the model tree, and store the associated model components within it. When your application grows in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



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

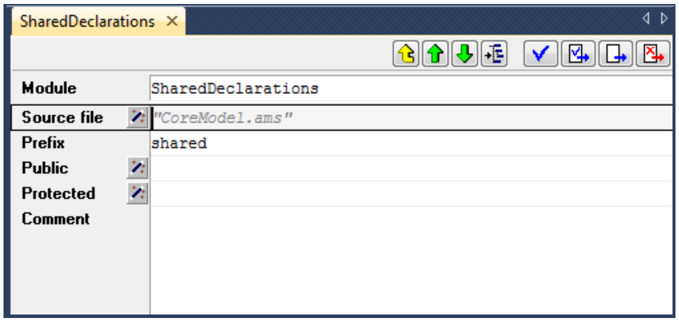
When you import the contents of a book section node into your model, you may find that particular identifier names in that book section already have been declared in the remainder of your model. If such a name clash occurs, AIMMS will refuse to import the specified `.ams` file into your model, and present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

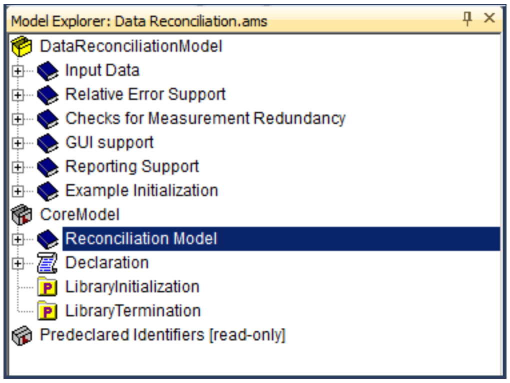
AIMMS uses modules to implement those parts of its functionality that can be best expressed in the AIMMS language itself. The available AIMMS system modules include

- 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™ 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

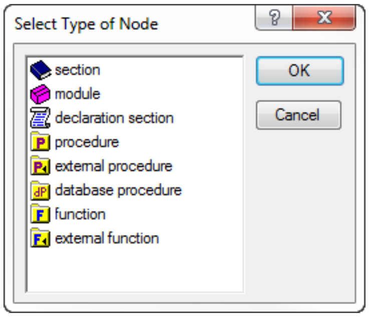
To create a new node in the model tree you must position the cursor at the node in the tree *after* which you want to insert a new node. You can create a new node here:

- 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.


Select Type of Node

- section
- module
- declaration section
- procedure
- external procedure
- database procedure
- function
- external function

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.