FICO™ Xpress Optimization Suite

Xpress IVE

User Manual

Release 1.24

Last update 30 March 2013

How to Contact the Xpress Team

Information, Sales and Licensing

USA, CANADA AND ALL AMERICAS

Email: XpressSalesUS@fico.com

WORLDWIDE

Email: XpressSalesUK@fico.com

Tel: +44 207 940 8718
Fax: +44 870 420 3601

Xpress Optimization, FICO
FICO House
International Square
Starley Way
Birmingham B37 7GN
UK

Product Support

Email: Support@fico.com
(Please include *'Xpress'* in the subject line)

Telephone:

NORTH AMERICA
Tel (toll free): +1 (877) 4FI-SUPP
Fax: +1 (402) 496-2224

EUROPE, MIDDLE EAST, AFRICA
Tel: +44 (0) 870-420-3777
UK (toll free): 0800-0152-153
South Africa (toll free): 0800-996-153
Fax: +44 (0) 870-420-3778

ASIA-PACIFIC, LATIN AMERICA, CARIBBEAN
Tel: +1 (415) 446-6185
Brazil (toll free): 0800-891-6146

For the latest news and Xpress software and documentation updates, please visit the Xpress website at http://www.fico.com/xpress or subscribe to our mailing list.

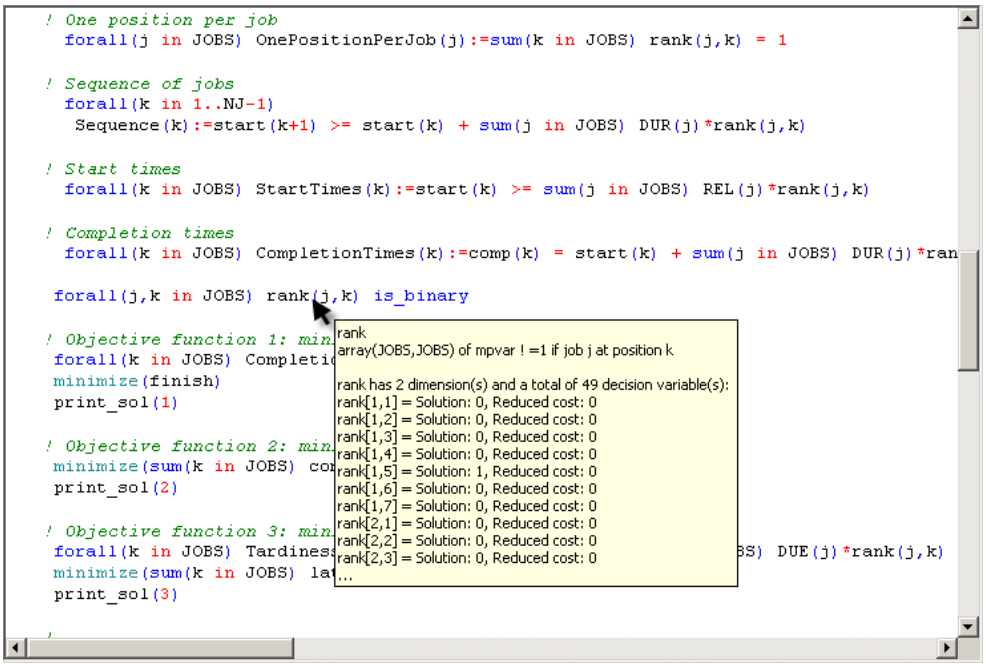
Contents

CHAPTER 1

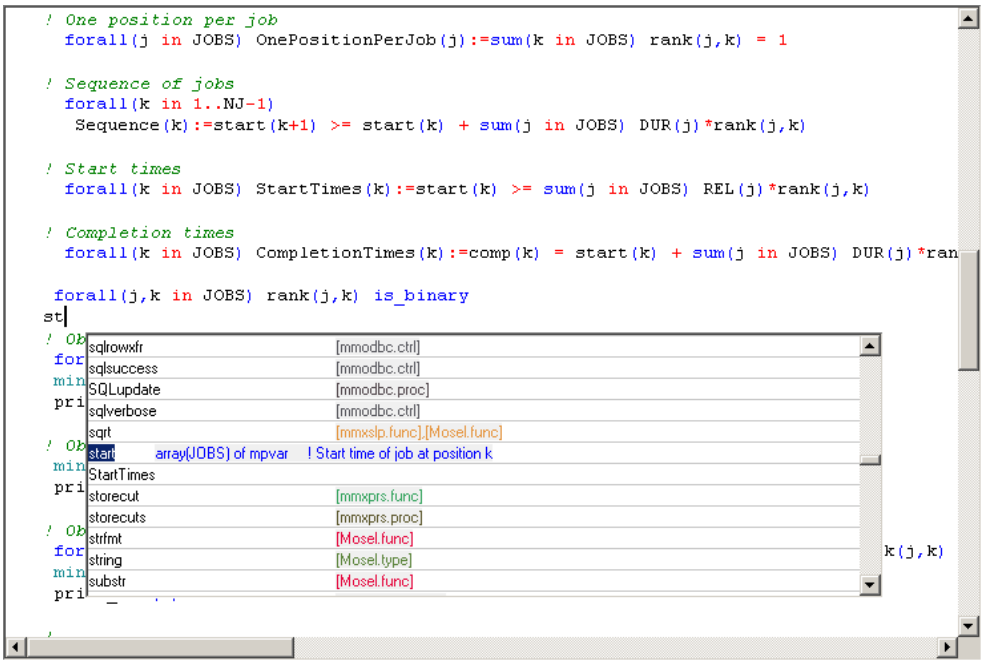
The Xpress-IVE Editor

The intelligent editor built into Xpress-IVE offers all the features of a modern programmer's text editor, plus enhancements designed to simplify working with Mosel models or LP and MPS files.

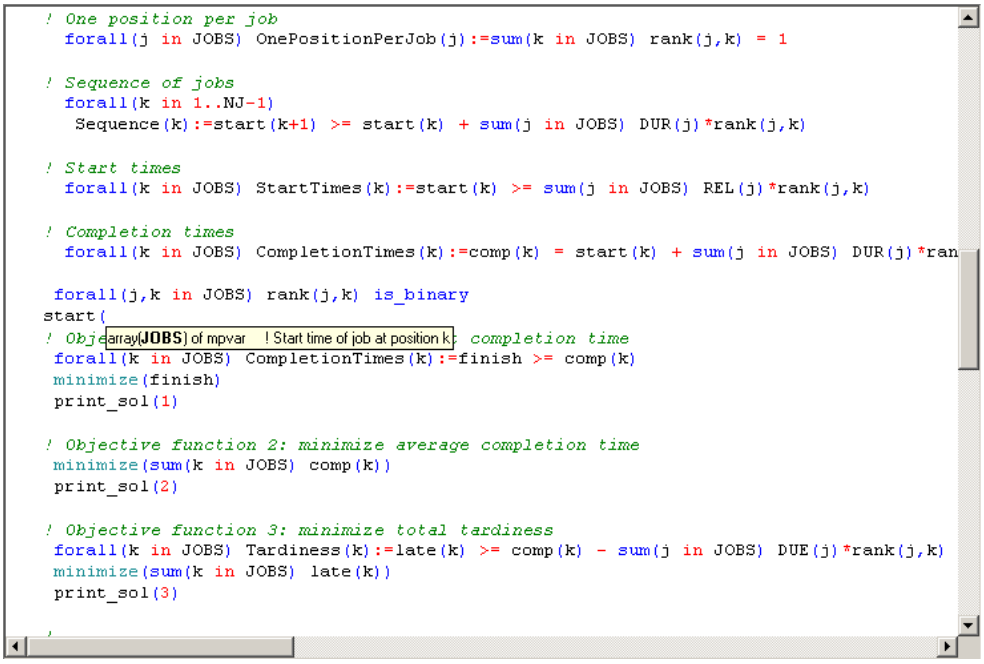
- The editor supports Mosel language *syntax highlighting*
- Two types of highlighting denote:
  1. Mosel language keywords;
  2. Identifier names defined in Mosel extension modules (dso's)
- *Mouse over information* on most identifiers in a model:



- *Auto-complete* feature: Press CTRL+Space to obtain a list of Mosel keywords and other identifier names that can be inserted at the current location:



- When pressing CTRL+Space as a function, procedure or array name is typed in, the signature (list of parameters or index sets) will appear in a tooltip, highlighting each parameter as it gets typed:

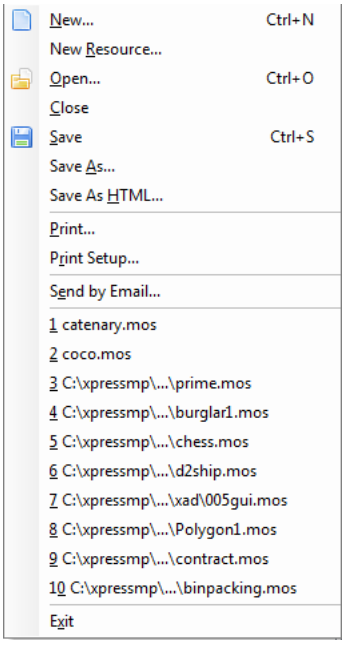


- *Editor settings* are available by *right-clicking* in the editor window and selecting **Properties...**

CHAPTER 2

Xpress-IVE Menus

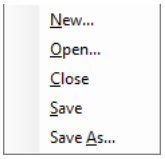
2.1 The File menu



New...	Create a new file.
New Resource...	Create a new resource file.
Open...	Open an existing file in the editor.
Close	Close the file currently shown in the editor.
Save	Save the file currently shown in the editor.
Save As...	Save the file currently shown in the editor under a different name.
Save As HTML...	Save the current file in HTML format, preserving formatting and colors. Example The HTML code can be copied to Microsoft Word or other rich text format editor, for inclusion in documentation, papers, etc.
Print...	Print the current file.

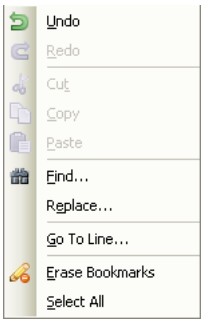
Print Setup...	Change printer settings.
Send by Email...	Send the current file by email if a registered MAPI handler exists.
Recently used files	Open one of the ten most recently used files.
Exit	Quit Xpress-IVE.

2.2 The Project menu



New...	Create a new Project.
Open...	Open an existing Project.
Close	Close the currently open Project.
Save	Save the currently open Project and any modified files contained within it.
Save As...	Save the currently open Project, under a different name, and any modified files contained within it. (Note that if you save the project file to a different directory, you will receive a warning that any files with relative paths will not be updated relative to the project's new location.) For more information on use of Projects in Xpress-IVE, see the Help entry on the Project Explorer Bar.

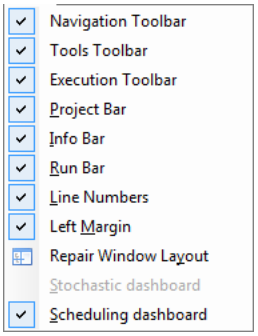
2.3 The Edit menu



Undo	Revert the editor to the state before the last editing action.
Redo	Cancel the previous Undo operation.
Cut	Cut the selected text and place it in the clipboard.
Copy	Place a copy of the selected text in the clipboard.

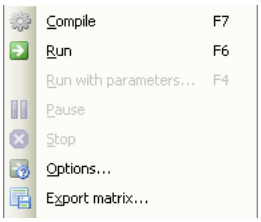
Paste	Insert the contents of the clipboard at the current location.
Find...	Open the search dialog.
Replace...	Open the replace dialog.
Go To Line...	Go to any line number in the editor.
Erase bookmarks	Remove any bookmarks placed by a recent search.
Select All	Select the entire text buffer in the editor.

2.4 The View menu



Navigation Toolbar	Show/hide the toolbar showing navigation buttons.
Execution Toolbar	Show/hide the toolbar showing execution buttons.
Tools Toolbar	Show/hide the toolbar showing debug tools buttons.
Model Explorer Bar	Show/hide the Model Explorer Bar.
Project Explorer Bar	Show/hide the Project Explorer Bar.
Info Bar	Show/hide the Info Bar.
Run Bar	Show/hide the Run Bar.
Line Numbers	Show/hide line numbering in the editor.
Left Margin	Show/hide a margin on the left side of the editor. Bookmarks will appear here.
Repair Window Layout	In case the bars in IVE are positioned wrongly or lost, select to reset the layout to the shipping default
Stochastic dashboard	Show/hide the Stochastic modeling dashboard when available.
Scheduling dashboard	Show/hide the Scheduling dashboard when available.

2.5 The Build menu



Compile Saves and compiles the current .mos (Mosel) file. If there are compilation errors, they will appear in the Info Bar. If compilation is successful, the entity tree will be populated with entities from the model.

Run Saves, compiles and executes the current .mos (Mosel) model. During the execution, the output from Mosel will go to the Output pane from the Run Bar. After the run terminates the entity tree will show actual values taken by variables in the Mosel model.

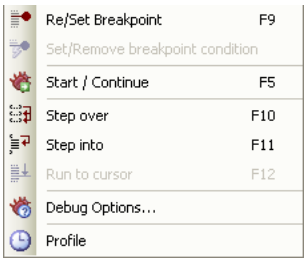
Pause Uncheck this option to resume the execution after it has been paused.

Stop Attempt to interrupt the execution of a model or an optimization by requesting that Mosel and the Optimizer stop. Mosel/the Optimizer will stop whenever it is safe to do so.

Options... Shows the Run options dialog, allowing the user to change settings for Mosel or Optimizer runs. It can also be invoked while a Mosel model is executing or during the optimization of a matrix file. Changes will take effect immediately after the dialog is dismissed.

Export matrix... Shows the Export matrix dialog for exporting matrices produced during Mosel runs to LP or MPS files.

2.6 The Debug menu

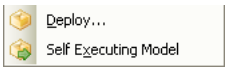


Re/Set Breakpoint Sets or removes a breakpoint on the current line.

Set breakpoint condition Invokes the Breakpoint condition dialog.

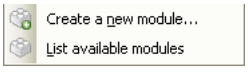
Start/Continue	Saves, compiles and begins/resumes debugging the current .mos (Mosel) model. While debugging, the execution will stop at breakpoints or can be controlled manually by stepping into the code. When the execution is interrupted, values for all identifiers (including those with a local scope) are available. Also, the Debug watches in the Info bar are updated.
Step over	While debugging, advance the execution point by one line, stepping over functions/procedure calls.
Step into	While debugging, advance the execution point by one line, stepping into functions/procedures.
Run to cursor	While debugging, run to the line where the editing cursor is positioned.
Debug Options...	Shows the Debug Options dialog for modifying the behavior of the debugger.
Profile	Saves, compiles and begins running the current .mos (Mosel) model. Every monitoring feature in IVE will be turned off to eliminate overheads; statistics will be gathered regarding time spent on every line of Mosel code. At the end of the run, the profiler output will be produced.

2.7 The Deploy menu



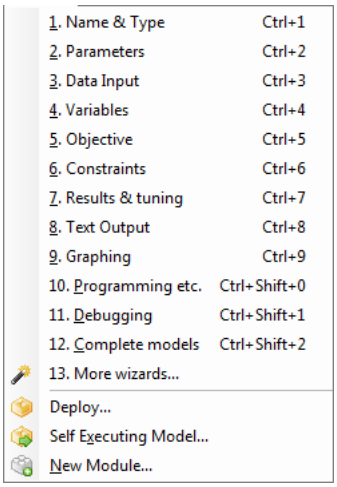
Deploy...	Opens the Deploy dialog.
Self-Executing Model	Opens the Self-Executing Model dialog.

2.8 The Modules menu



Create a new module	Opens the New Module Wizard dialog.
List available modules	Opens the List Modules dialog.

2.9 The Wizards menu

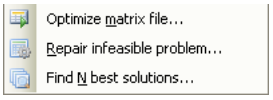


1. Name & Type Wizard	Invokes the Name & Type Wizard
2. Parameters Wizard	Invokes the Parameters Wizard
3. Data Input Wizard	Invokes the Data Input Wizard
4. Variables Wizard	Invokes the Variables Wizard
5. Objective Wizard	Invokes the Objective Wizard
6. Constraints Wizard	Invokes the Constraints Wizard
7. Results & tuning Wizard	Invokes the Results & tuning Wizard
8. Text Output Wizard	Invokes the Text Output Wizard
9. Graphing Wizard	Invokes the Graphing Wizard
10. Programming Wizard	Invokes the Programming Wizard
11. Debugging Wizard	Invokes the Debugging Wizard
12. Complete models Wizard	Invokes the Complete models Wizard
13. More wizards...	Invokes the Wizard viewer without selecting any particular wizard
Deploy...	Opens the Deploy dialog.
Self-Executing Model	Opens the Self-Executing Model dialog.
Create a new module	Opens the New Module Wizard dialog.

2.10 The Window menu

Cascade	Cascade the opened windows.
Tile	Tile the opened windows.
Arange Icons	Arrange icons when all windows are minimized.

2.11 The Optimizer menu

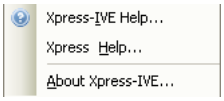


Optimize matrix file... Shows the Optimizer dialog which allows for the optimization of matrices in LP or MPS format.

Repair infeasible problem... Shows the Optimizer dialog with options for relaxing an infeasible problem.

Find N best solutions... Shows the Optimizer dialog which options for finding alternate optimal solutions.

2.12 The Help menu



Xpress-IVE Help...	Opens this help system.
Xpress Help...	Opens the comprehensive Xpress Help system which includes in-depth documentation on all Xpress products.
About Xpress-IVE...	Shows the About dialog, including version and copyright information.

CHAPTER 3

Xpress-IVE Toolbars

The Bars display rich information on the status of Xpress-IVE, the degree of success in compiling/running Mosel models, and results from runs. In addition, the Bars simplify user interaction with the application by placing most controls within easy reach. The Bars can be resized or repositioned—settings are always saved on exit.

3.1 The Toolbars

3.1.1 Navigation Toolbar

Create a new file and open it in the editor. (also present in *File* menu)

Open an existing file in the editor. (also present in *File* menu)

Save the file currently shown in the editor. (also present in *File* menu)

Cut the selected text and place it in the clipboard. (also present in *Edit* menu)

Place a copy of the selected text in the clipboard. (also present in *Edit* menu)

Insert the contents of the clipboard at the current location. (also present in *Edit* menu)

Go to the previous position in the editor (also present in *View* menu)

Go to the next position in the editor (also present in *View* menu)

Go to the previous indentation level (also present in *View* menu)

Go to the next indentation level (also present in *View* menu)

Undo last text editor action (also present in *View* menu)

Redo last text editor action (also present in *View* menu)

Remove any bookmarks placed by a recent search. (also present in *Edit* menu)

Search within the text editor. (also present in *Edit* menu)

Repair Window Layout (also present in *View* menu)

Show the Xpress-IVE help system.

3.1.2 Execution Toolbar

Show the Wizards Dialog:

- Show the 1. Name & Type wizard. (also present in *Wizards* menu)
- Show the 2. Parameters wizard. (also present in *Wizards* menu)
- Show the 3. Data Input wizard. (also present in *Wizards* menu)
- Show the 4. Variables wizard. (also present in *Wizards* menu)
- Show the 5. Objective wizard. (also present in *Wizards* menu)
- Show the 6. Constraints wizard. (also present in *Wizards* menu)
- Show the 7. Results & tuning wizard. (also present in *Wizards* menu)
- Show the 8. Text Output wizard. (also present in *Wizards* menu)
- Show the 9. Graphing wizard. (also present in *Wizards* menu)
- Show the 10. Programming wizard. (also present in *Wizards* menu)
- Show the 11. Debugging wizard. (also present in *Wizards* menu)
- Show the 12. Complete models wizard. (also present in *Wizards* menu)

Show the Deploy dialog. (also present in *Deploy* menu)

Show the Self-Executing Model dialog. (also present in *Deploy* menu)

Save and compile the current model. (also present in *Build* menu)

Show the Run options dialog. (also present in *Build* menu)

Save, compile and run current model. (also present in *Build* menu)

Uncheck to resume the execution after it has been paused. (also present in *Build* menu)

Interrupt the execution. (also present in *Build* menu)

Show the Export matrix dialog. (also present in *Build* menu)

Show the Optimizer dialog. (also present in *Optimizer* menu)

Show the Optimizer dialog with options to repair infeasible problems. (also present in *Optimizer* menu)

Show the Optimizer dialog with options to find alternate optimum solutions. (also present in *Optimizer* menu)

Show the List Modules dialog. (also present in *Modules* menu)

Show the New Module Wizard dialog. (also present in *Modules* menu)

3.1.3 Tools Toolbar

 Re/Set breakpoint. (also present in *Debug* menu)

 Set/Remove breakpoint condition by invoking the Breakpoint condition dialog. (also present in Debug menu)

 Start/Continue debugging. (also present in *Debug* menu)

 Step over. (also present in *Debug* menu)

 Step into. (also present in *Debug* menu)

 Run to cursor. (also present in *Debug* menu)

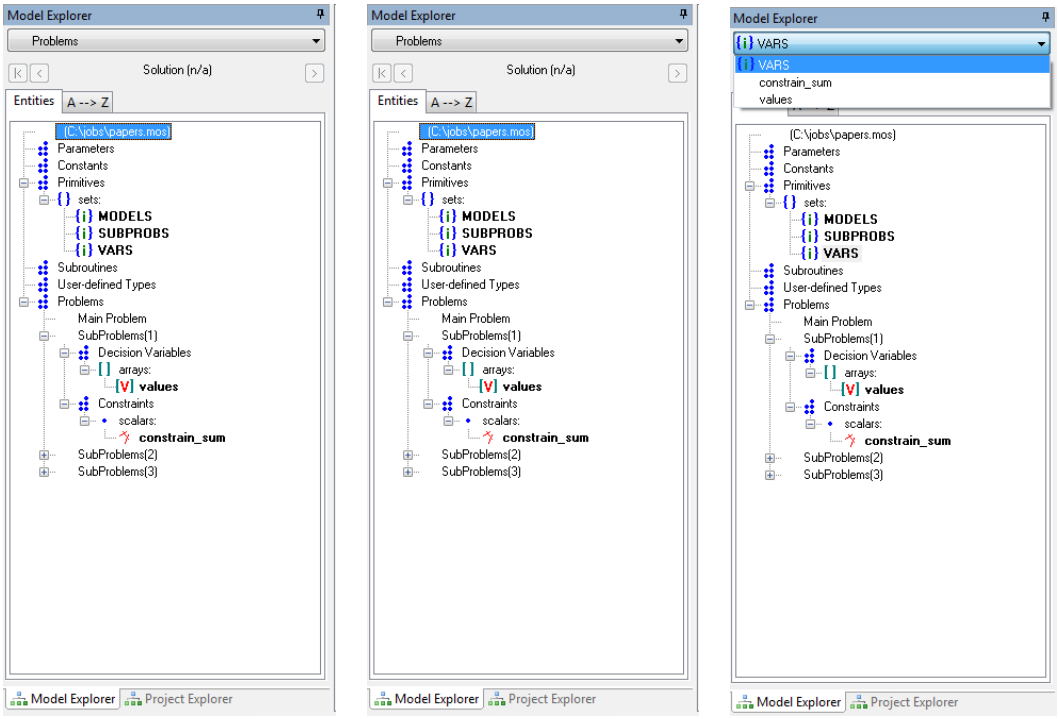
 Invokes the Debug Options dialog. (also present in *Debug* menu)

 Start profiling the Mosel model. (also present in *Debug* menu)

3.2 The Model Explorer Bar showing the entity tree

Can be shown/hidden by selecting *Model Explorer Bar* from the *View* menu or by clicking the  icon on the Project pane.

The Model Explorer Bar contains the entity tree, the A->Z entity list and the most recently accessed entities combo list:



After the successful compilation of a Mosel model, the entity tree will be populated with identifiers from the model. After a successful execution of the model, the values of the identifiers

appear in tooltips when the mouse is nearby. The complete values can be examined by double clicking on an identifier (which will open the View text dialog). If a model is being solved for multiple solutions, then the controls under recently-accessed entities list can be used to select a given solution (the best solution is chosen by default).

The identifiers are grouped in the following categories:

Parameters
Parameters can have the same type as Mosel primitives, but they are special entities as they can be used to pass information to a model without having to edit it.

Constants
Constants also have regular primitive types, but their values cannot change at runtime.

Primitives
These are primitive types (integer, real, boolean, string) either standalone or grouped into sets or arrays.
User types (defined within Mosel modules) will also appear here. Refer to the Mosel Native Interface documentation for details on how to define and use such external types.
Note that if user types do not support conversion to/from text, the values shown here will be meaningless.

Subroutines
A list of all the functions and procedures defined in the current model.

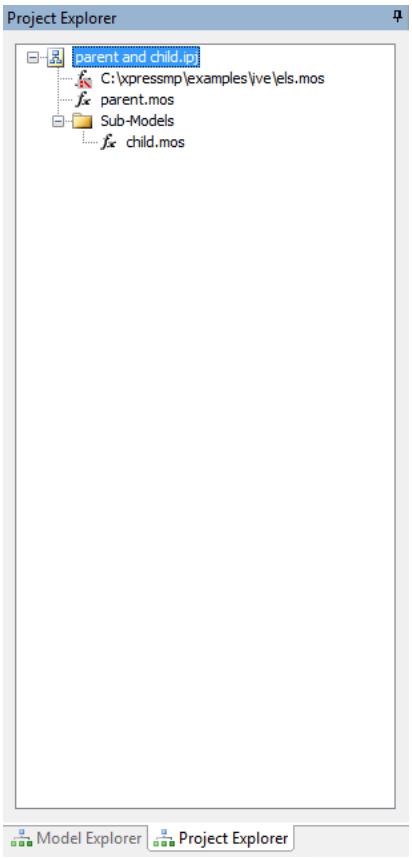
Problems
The main Mosel problem, and any other variables containing problem objects (which can be selected in model by use of the Mosel with statement). In the example images above, the model contains an variable called SubProblems, an array of 3 mpproblem objects.

Those entities which are displayed specific to a given problem are:
Decision variables
Representing the special mpvar Mosel type, decision variables are characterized by solution value and reduced cost after a successful optimization.
Linear constraints
Representing the special linctr Mosel type, linear constraints are characterized by activity value, slack and dual value after a successful optimization.

Note that decision variables and linear constraints will only display correct values if the optimization was successful and a solution was found.

3.3 The Project Explorer Bar

Can be shown/hidden by selecting *Project Explorer Bar* from the *View* menu or by clicking the  icon on the Project pane. Creation, loading, and saving of Project files is handled by the Project menu.



Projects in IVE are a collection of files (which can be any type, not just Mosel or XAD resource files), along with the options specified in the Run Options dialog. One can classify the files in a Project into a hierarchy of folders (not be confused with file system directories). Thus you might wish to arrange files in a Project according to purpose, for example:

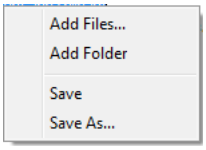
- sub-models in one folder, and the main models that use them in another (or at the top level);
- a folder for each file type (as is the default behaviour for projects in Microsoft Visual Studio);
- folder layout to mirror the hierarchy of files on disk.

Files in a project will be stored as either absolute or relative paths. If a file in a project is in same directory as the project itself, or one of its subdirectories, its path will be stored relative to the project file's directory. A file from outside this directory hierarchy will be stored as an absolute path. In the example image above, the files parent.mos and child.mos are stored in the same directory as the project, whereas the file els.mos is in an unrelated directory and so is stored as an absolute path. Files with absolute paths have the stylised colon-backslash symbol overlaid on their icons in the project tree.

3.3.1 Operations on Projects

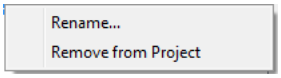
There are a number of actions one can perform with a Project. Many of these are started by bringing up a context-menu, typically by right-clicking the mouse on an item in the project tree.

3.3.2 Pop-up menu for the Project



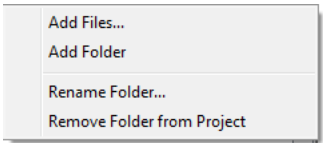
Add Files...	This brings up a file dialog box allowing the user to add files to the top level of the Project
AddFolder	This creates a new empty Folder (as a direct child of the Project), which can then be renamed and populated.
Save	Same as the Save action on the Project menu. (When you save a Project, all modified files within it are also saved).
Save As...	Same as the Save As action on the Project menu. (When you save a Project, all modified files within it are also saved).

3.3.3 Pop-up menu for files



Rename	This makes the file name editable. See Editing file and folder names below. This can also be acheived by selecting the item in the Project tree and pressing F2.
Remove from Project	This removes the file from the project, but does not delete it from disk. This can also be acheived by selecting the item in the Project tree and pressing the Delete key.

3.3.4 Pop-up menu for folders



Add Files...	This brings up a file dialog box allowing the user to add files to this folder.
AddFolder	This creates a new empty folder (as a direct child of this folder), which can then be renamed and populated.
Rename Folder	This makes the folder name editable. See Editing file and folder names below. This can also be acheived by selecting the item in the Project tree and pressing F2.
Remove Folder from Project	This removes the folder (and any contents) from the project, but does not delete any files from disk. This can also be acheived by selecting the item in the Project tree and pressing the Delete key.

3.3.5 Drag-and-Drop within the Project tree

In order to rearrange the structure of the Project hierarchy, files and folders can be moved by Drag-and-Drop with the mouse. For instance, when a new folder has been created, other files or folders can be dragged into it.

3.3.6 Editing file and folder names

When a file or folder is made editable, a text box is brought up over its location with the current contents selected. Having adjusted the name, press the Enter key to confirm the change, or the Escape key to cancel. If you are renaming a file that is open in the IVE editor, the file will also be renamed in the editor. Note: you cannot use this method to move files to different directories, merely to rename files with a directory; this is why only the file name is editable, not the whole path.

3.3.7 Run options

When a Project is saved, the contents of the Run options dialog are saved with it. These options are then restored when the Project is reloaded. This facilitates keeping files together with their relevant settings.

3.3.8 The PROJECTDIR parameter

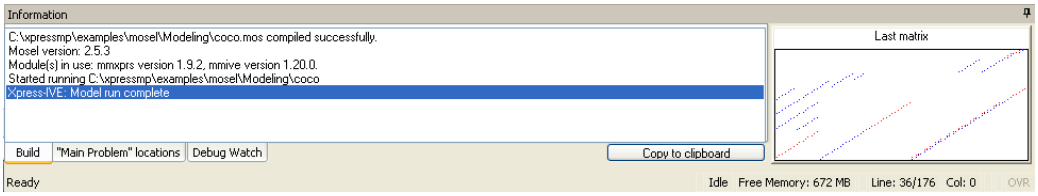
If a Project is laoded when a model is run, then a Mosel parameter named PROJECTDIR is set to the full path of directory in which the poject file is located. This value can be accessed in the usual way via a Mosel parameters...end-parameters block. This value can be seen in the Run options dialog. (If the users sets a Mosel parameter PROJECTDIR via the Run options dialog, this will override the Project-supplied version).

3.4 Info Bar

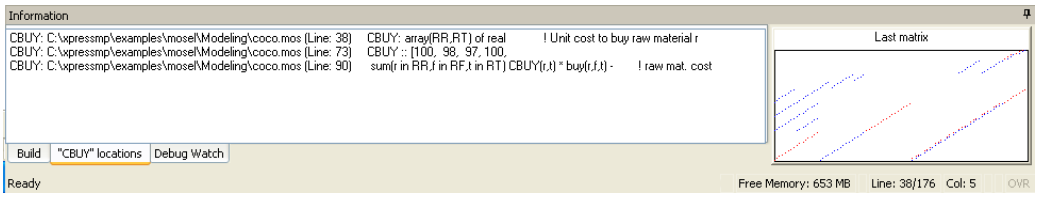
Can be shown/hidden by selecting *Info Bar* from the *View* menu or by clicking the icon on the Information pane.

The Info Bar consists of three views:

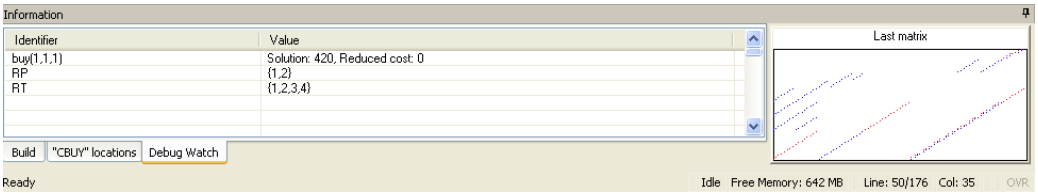
1. Build — Shows the status of the compilation of a Mosel model and reports errors if any. Click on an error to make the editor navigate to the offending line in the model. When running a model, some brief statistics and version information appear here. Press the button Copy to clipboard to copy the contents of the Build view to the Windows clipboard. Then paste into any text editor/email client.



2. "..." locations — When an entity is clicked on in the entity tree, all the lines where the entity is present are added to this clickable list. Click on a line in the list to highlight it in the editor.



3. Debug watch — Editable list of identifiers whose value is shown.



In addition, the Info Bar shows a sketch of the latest matrix loaded in the Optimizer. Note that the matrix may be "presolved".

3.5 Tools Bar

Can be shown/hidden by selecting *Tools Bar* from the *View* menu or by clicking the  button on the Toolbar.

The Tools Bar is made up of five regions:

- On the left, a list of buttons corresponds to the files opened in the editor. Switch from one file to another simply by clicking the button having the chosen file name.
- The  (Ctrl+Alt+Left) and  (Ctrl+Alt+Right) buttons navigate through current and previous locations in the editor.
- The  (Ctrl+Alt+Up) and  (Ctrl+Alt+Down) buttons find a previous or next line in the editor with the same indentation level as the current line.
- The *Find* button (marked by binoculars) opens the search dialog.
  - The search box can be typed in or pasted to from the clipboard. Press Enter to repeatedly search in the current file for occurrences of the word or expression in the search box.
  - The small progress bar below the search box shows the position of the editing cursor in the current file.
- The context *Help* button – if pressed, it will attempt to find Help on the current word in the editor.

3.6 Run Bar

Can be shown/hidden by selecting *Run Bar* from the *View* menu or by clicking the  icon on the Run pane.

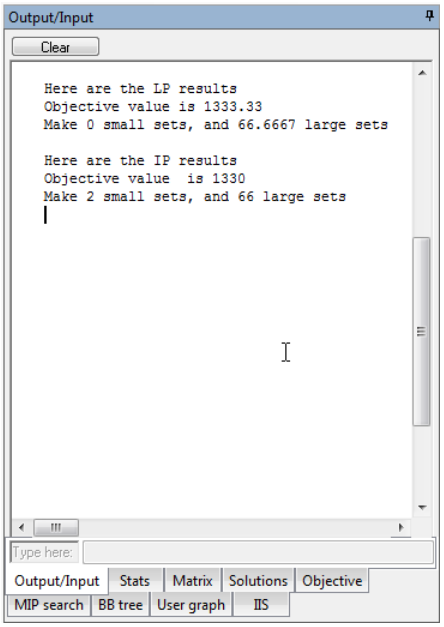
The Run Bar groups together the following tabs/panes: Output/Input, Stats, Matrix, Solutions, Objective, MIP search, BB tree, SLP progress, User graph, CP stats, and CP search.

The Run Bar can be put into space-saving mode by enabling the *Auto hide* checkbox: when the mouse leaves the area of the Run Bar and enters the editor, the Run Bar will be hidden almost completely. When the mouse re-enters the Run Bar, it expands to its previous dimensions.

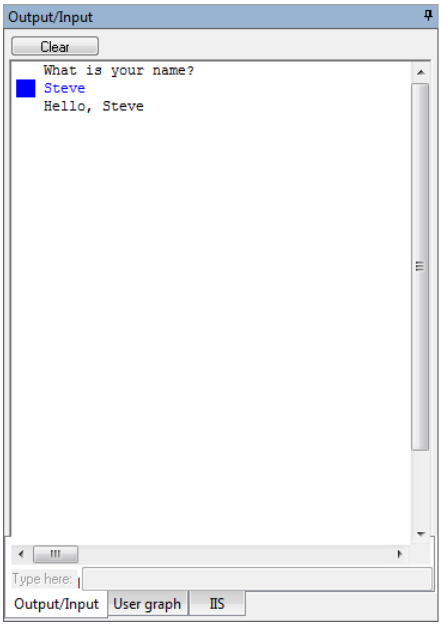
Below are snapshots of the different tabs/panes:

3.6.1 Output/Input

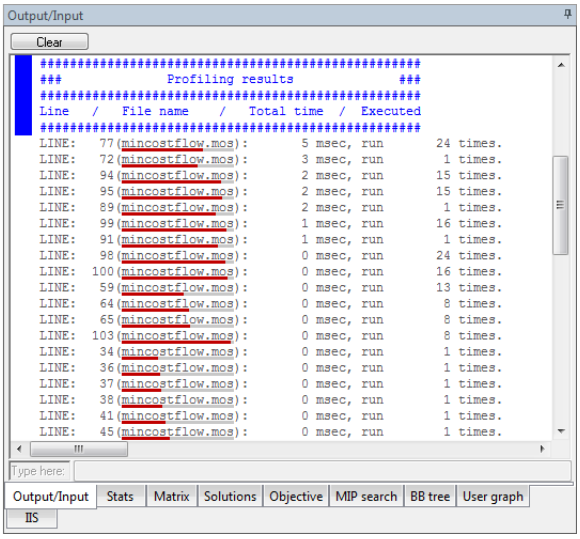
Mosel output



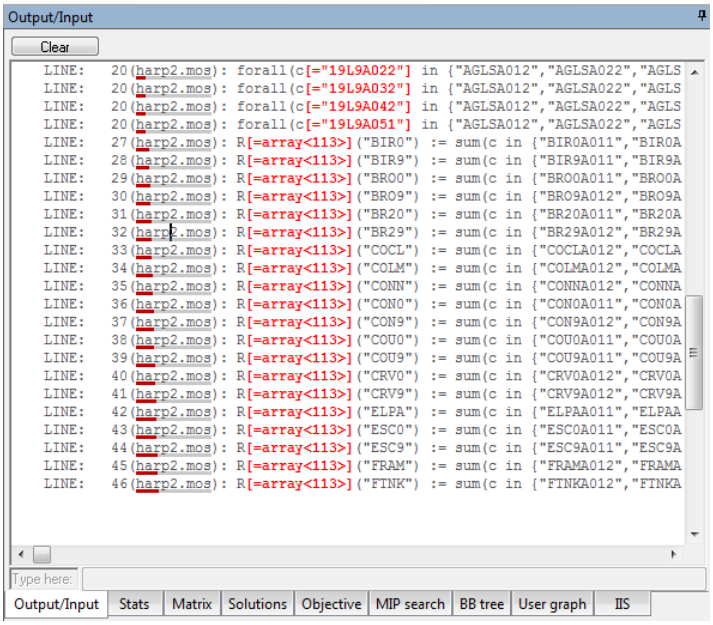
Console-style user input:



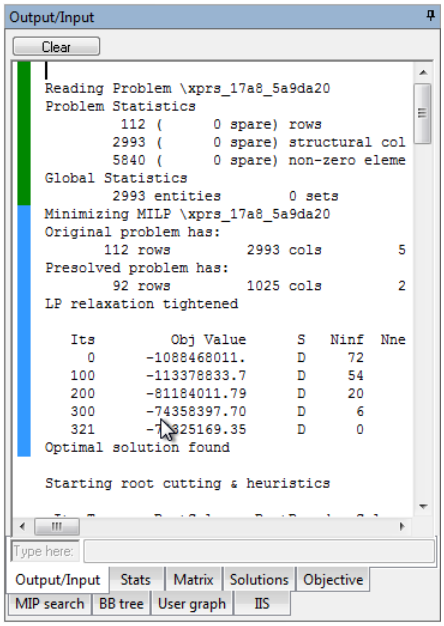
Profiler output:



Debugger output:

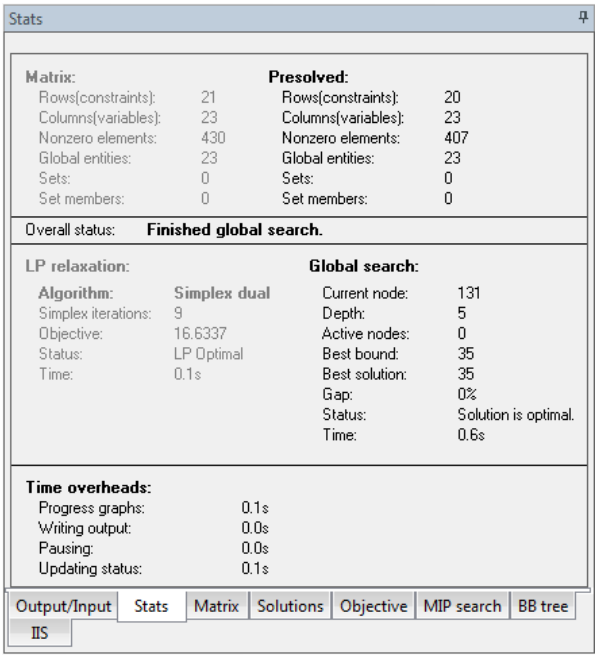


Optimizer output when a matrix file is run using the Optimizer Dialog:



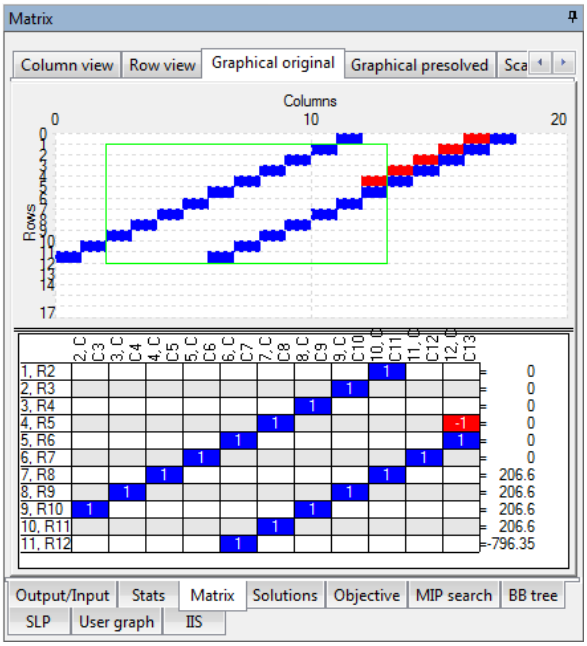
3.6.2 Statistics

Optimizer status and statistics:



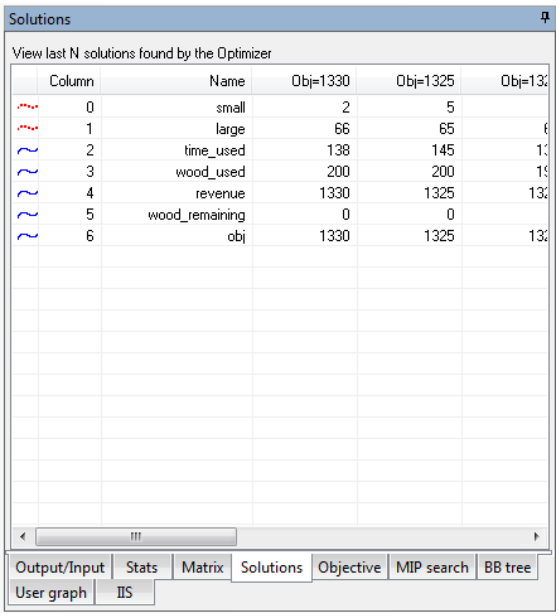
3.6.3 Matrix

A zoomable view of the matrix:



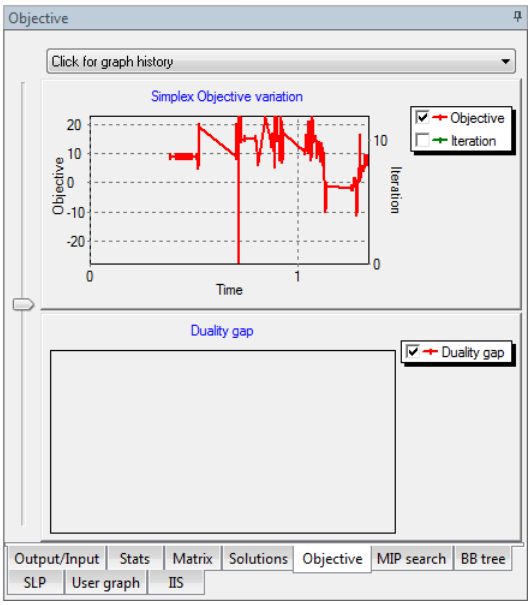
3.6.4 Solutions

The last N solutions found by the Optimizer:



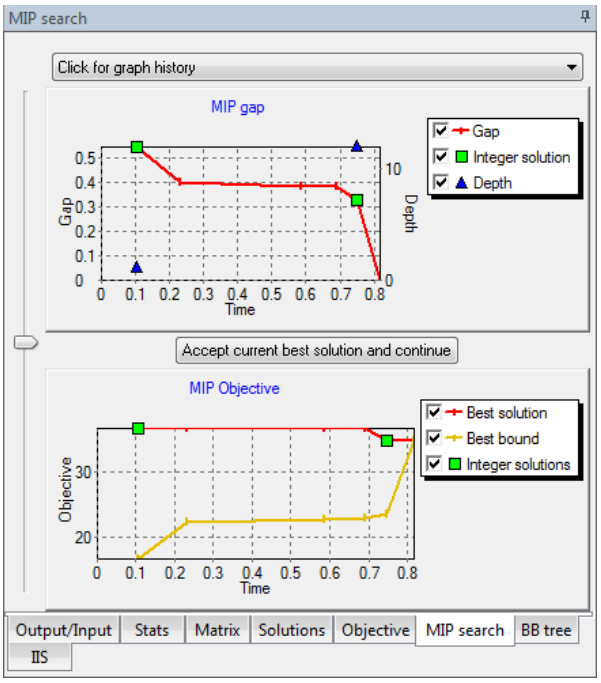
3.6.5 Objective

Objective value progress during Simplex or Newton barrier:



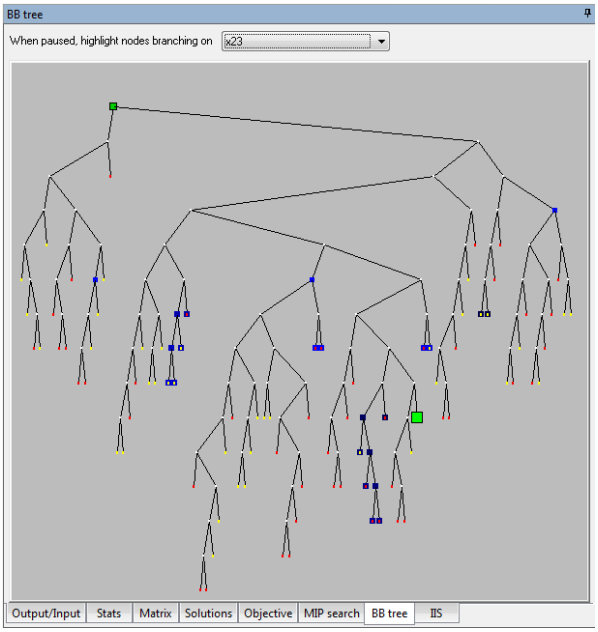
3.6.6 MIP search

Progress of the global search for integer solutions:



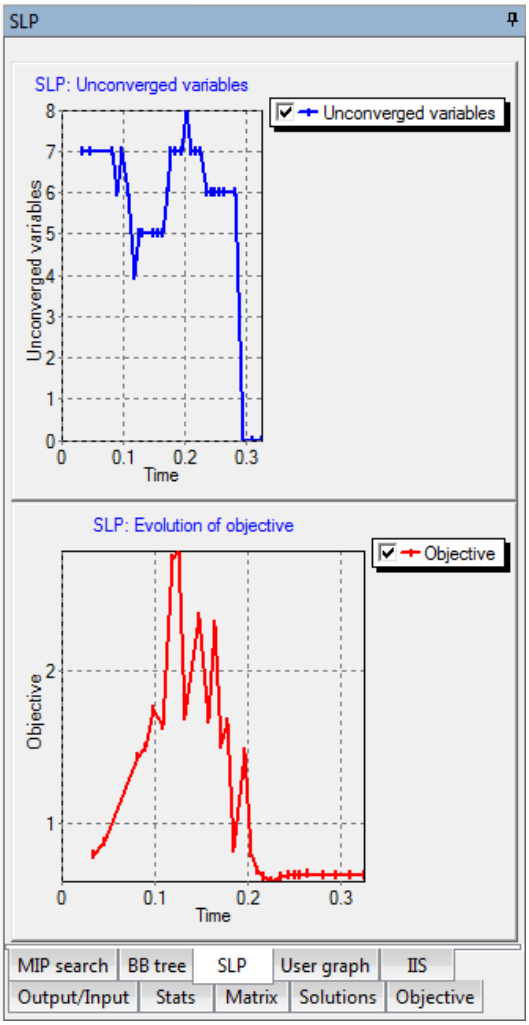
3.6.7 BB tree

The evolution of the Branch and Bound MIP search:



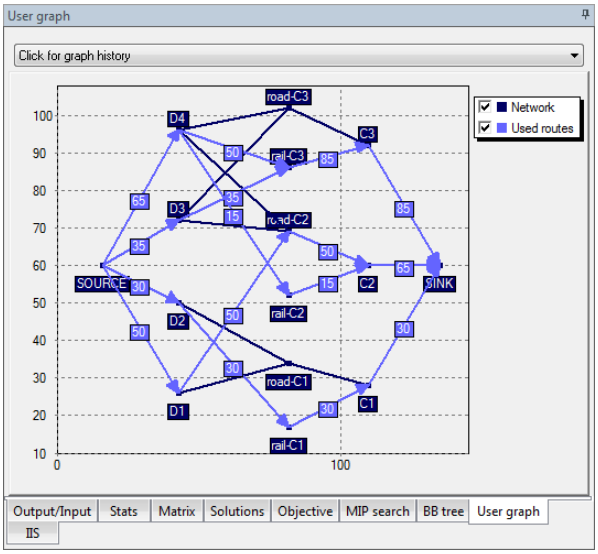
3.6.8 SLP progress

SLP progress:



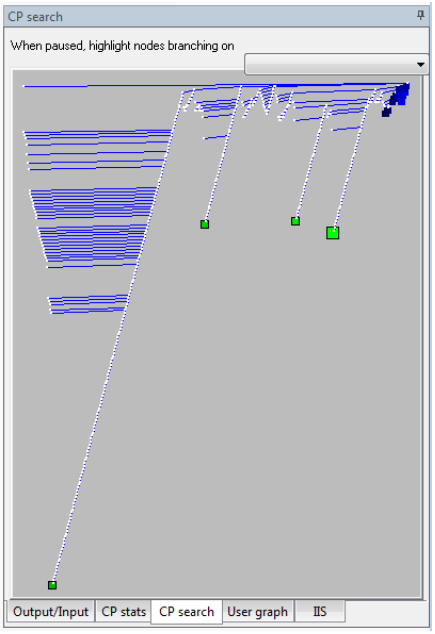
3.6.9 User graph

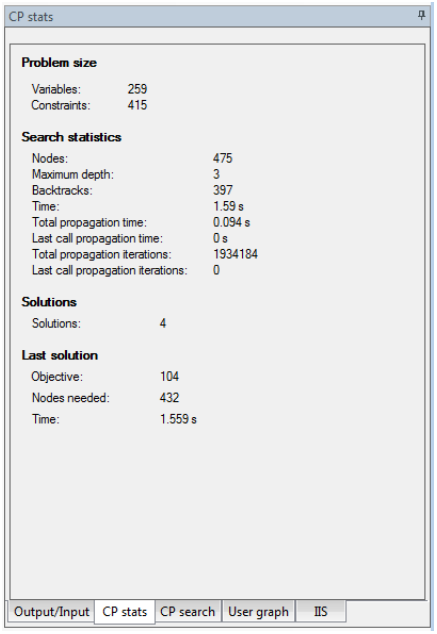
Graph constructed by user during the Mosel run:



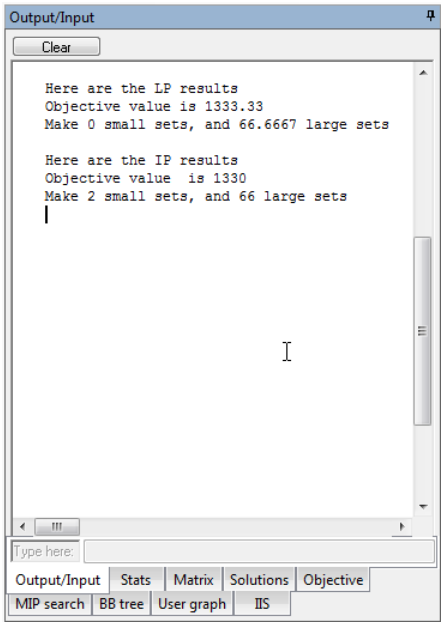
3.6.10 CP stats and CP search

Models using the Xpress-Kalis solver generate display in two additional tabs:





3.7 The Output/Input tab of the Run Bar



All write and writeln statements in Mosel produce character output that can be viewed in this window. Output from the Xpress Optimizer can also be shown by setting the boolean "mmxprs" parameter XPRS_VERBOSE.

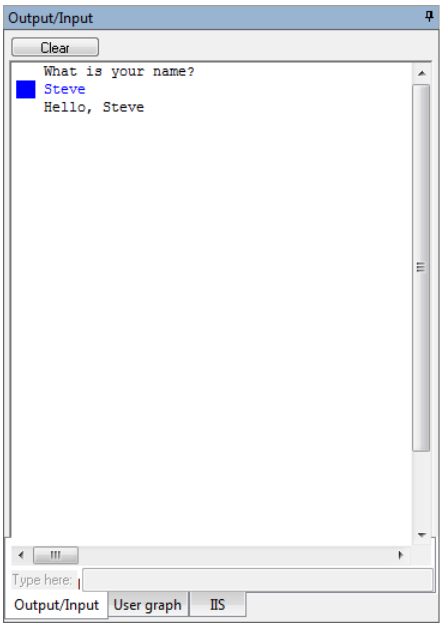
Notes:

- To scroll the *Output* window using the keyboard, click once on the window to obtain the

keyboard focus.

- Text can be selected and copied directly from the *Output* window.
- To clear the contents of the *Output* window, click on the *Clear* button.

3.7.1 User input in the Output/Input tab of the Run Bar

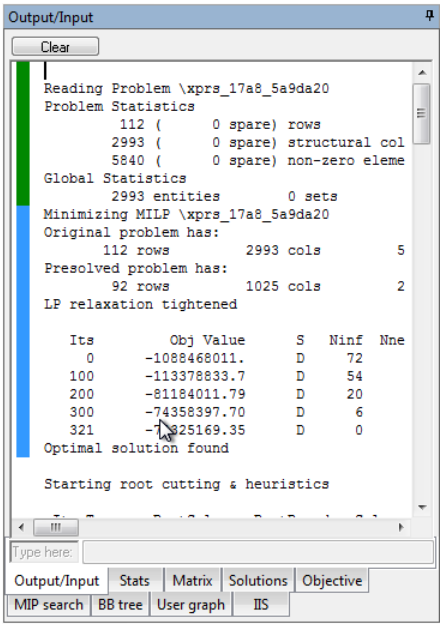


When the user is required to input text during a Mosel run (when the `read` or `readln` statements are executed), the text can typed in the edit box marked Type here. (This edit box is disabled except when expecting user input).

Notes:

- To send the typed text to Mosel, press Enter.
- After text has been input, the left margin of the corresponding line from the *Output* window will be highlighted in blue.
- Call the Mosel procedure `fflush` to display all remaining text before asking for input.

3.7.2 The Optimizer Output tab of the Run Bar

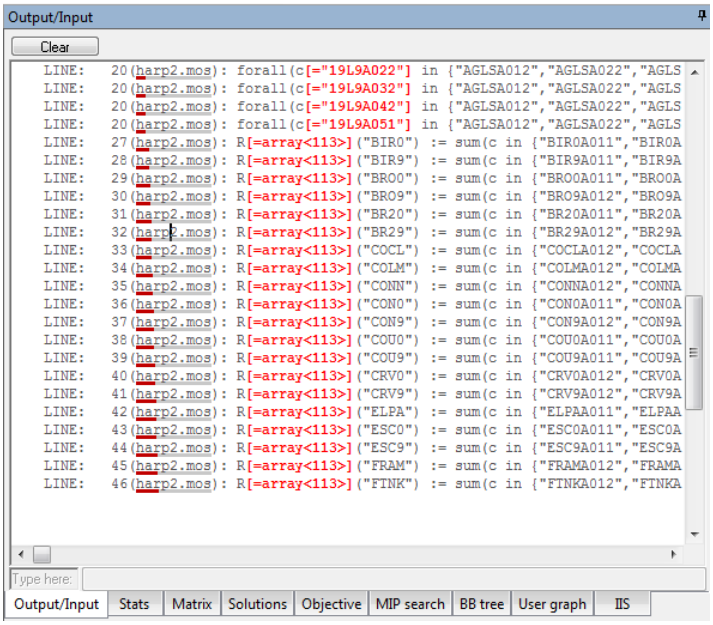


During a standalone Optimizer run (using the Optimizer dialog) or in the solving stage during a Mosel run (if the "mmxprs" parameter XPRS_VERBOSE was set to true), a textual Optimizer progress report will be produced in the Output window.

A typical output sequence from the Optimizer contains:

1. A short version and copyright message from the Optimizer, marked with a blue margin (does not appear during a Mosel run).
2. Information on any control parameter settings, written in magenta (does not appear during a Mosel run).
3. The problem reading phase and statistics, marked with a dark green margin.
4. The LP relaxation phase (Simplex or Barrier), marked with a light blue margin.
5. The global search phase (if applicable), marked with an orange margin.
6. The nonzero values in the solution vector, if a solution exists (does not appear during a Mosel run). These values are printed on alternating background colors to improve readability. There is no relationship between the background color and the value shown.
7. To learn how to control the amount and frequency of textual output from the Optimizer, please refer to the Xpress-Optimizer reference manual or the *7. Results & tuning* wizard.

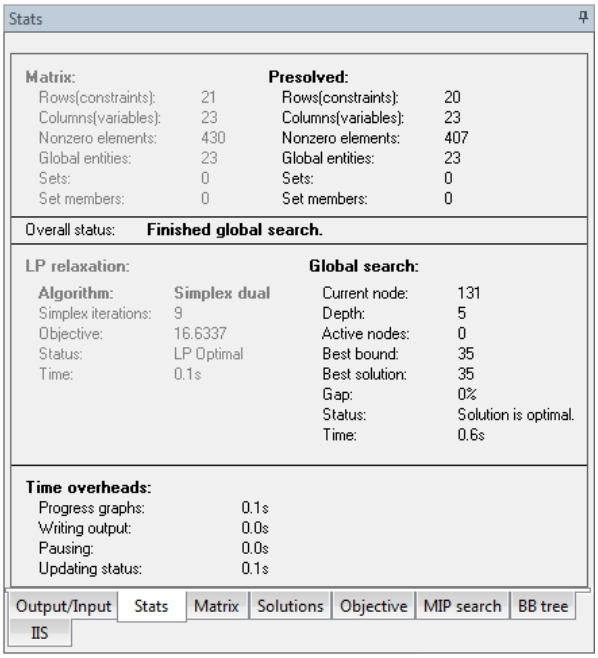
3.7.3 Debugger output in the Output/Input tab of the Run Bar



While debugging Mosel code, the user has the option of printing the current executed line. The line can also be expanded to show current values of primitive identifiers (integers, strings, booleans, reals) and current sizes of arrays and sets (see the *Debug Options* dialog for more information).

Clicking on a line number will activate the corresponding line in the editor.

3.8 The Stats tab of the Run Bar



During the optimization (solving) stage of a Mosel run, or when optimizing a matrix file, this

window pane displays the state of the Xpress Optimizer.

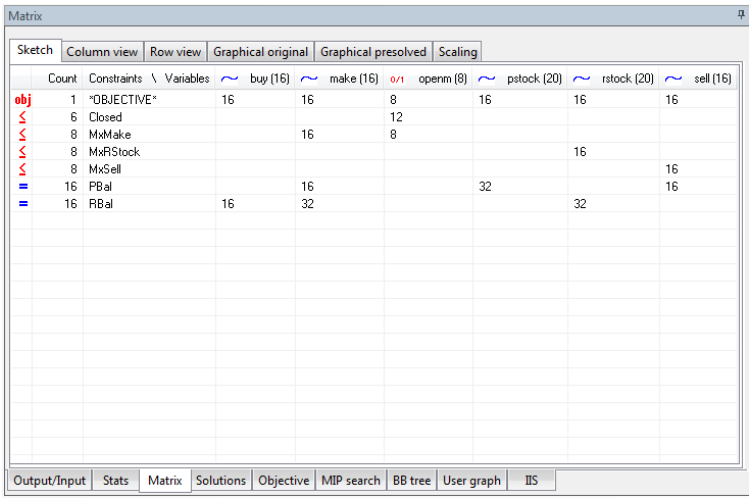
Text is highlighted (black) when the status or activity is current or currently taking place. Text is dimmed (gray) when the status information depicted is no longer current (*e.g.*, when the optimizer is processing a presolved matrix, the before-presolve matrix is not currently relevant) or if the activity described is either finished and still relevant (the LP relaxation statistics are useful during the global search phase) or not started yet.

The Time overheads section shows the approximate time spent by Xpress-IVE on tasks other than optimization. Note that the time overhead created by other programs (processes) running on the same machine is not accounted for, even though they can slow down the optimization significantly.

3.9 The Matrix tab of the Run Bar

- *Sketch view* : A summarized view of the rows and columns that make up the matrix
- *Column view* : A list of all columns in the matrix
- *Row view* : A list of all rows in the matrix
- *Graphical view* : An interactive graphical representation of the matrix contents
- *Scaling view* : A histogram showing matrix coefficient ranges

3.9.1 The Sketch View



This view attempts to summarize arrays of constraints and arrays of variables into logical units to give a more concise representation of the matrix in the Optimizer.
In the example above, the following information can be gleaned about the matrix from the sketch view:

1. There are 6 rows of type = which are named Closed.
2. There are 8 rows of type = which are named MxMake.
3. There are 8 rows of type = which are named MxRStock.
4. There are 8 rows of type = which are named MxSell.
5. There are 16 rows of type = which are named PBal.

6. There are 16 rows of type = which are named RBal.
7. In ALL the 6 = rows named Closed, variables named openm appear 12 times.
8. In ALL the 8 = rows named MxMake, variables named make appear 16 times.
9. In ALL the 8 = rows named MxMake, variables named openm appear 8 times.
10. In ALL the 16 = rows named PBal, variables named make appear 16 times.
11. In ALL the 16 = rows named PBal, variables named pstock appear 32 times.
12. In ALL the 16 = rows named PBal, variables named sell appear 16 times.
13. The objective function references variables named buy 16 times, variables named make 16 times, variables named openm 8 times, ...
14. and so on...

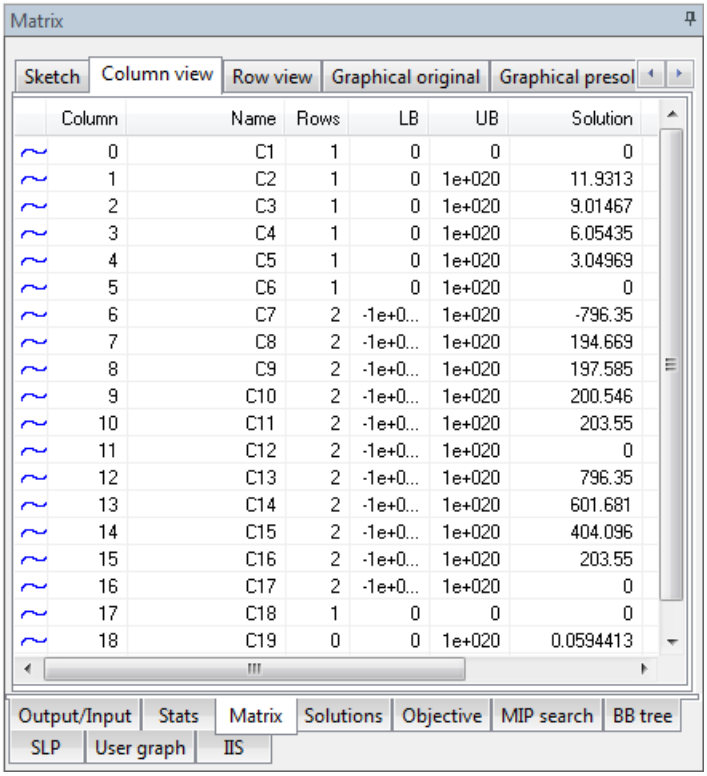
Notes:

- Only constraints and variables sent to the Optimizer will be represented.
- In most cases, by examining the names of variables and constraints in the matrix, IVE can determine their 'roots' and implicitly the logical arrays they belong to. Sometimes, however, the intentions of the user may not be guessed correctly. Please keep in mind these actions performed by IVE on each variable and constraint name to determine its root:

  1. All blanks at the beginning of the name are removed.
  2. All blanks at the end of the name are removed.
  3. The name is then searched from left to right for any of these letters: "{[(0123456789". When any such letter is found, that letter and all subsequent letters are thrown away.
  4. If the previous three steps produce non-empty name, it is the designated as the root of the original name.
  5. If the previous three steps yield an empty name, start again with the original name and do the following:
  6. Remove all digits ("0123456789") from the name.
  7. The remaining letters in the name are designated as the root.

- Some examples:
  The root of x[1,2] is x
  The root of production_level("Detroit","April",5) is production_level

3.9.2 The Column View



Column	Name	Rows	LB	UB	Solution
0	C1	1	0	0	0
1	C2	1	0	1e+020	11.9313
2	C3	1	0	1e+020	9.01467
3	C4	1	0	1e+020	6.05435
4	C5	1	0	1e+020	3.04969
5	C6	1	0	1e+020	0
6	C7	2	-1e+0...	1e+020	-796.35
7	C8	2	-1e+0...	1e+020	194.669
8	C9	2	-1e+0...	1e+020	197.585
9	C10	2	-1e+0...	1e+020	200.546
10	C11	2	-1e+0...	1e+020	203.55
11	C12	2	-1e+0...	1e+020	0
12	C13	2	-1e+0...	1e+020	796.35
13	C14	2	-1e+0...	1e+020	601.681
14	C15	2	-1e+0...	1e+020	404.096
15	C16	2	-1e+0...	1e+020	203.55
16	C17	2	-1e+0...	1e+020	0
17	C18	1	0	0	0
18	C19	0	0	1e+020	0.0594413

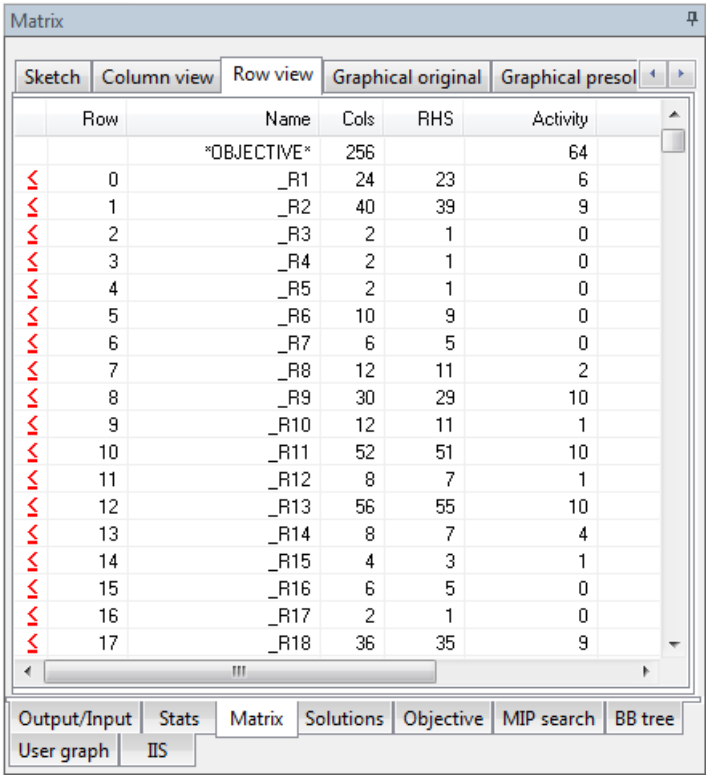
This view lists all variables (columns) in the matrix currently loaded in the Optimizer.
Note:

- Not all variables declared in the model may have been sent to the Optimizer!
- Values are updated as they become available (*e.g.*, if a new integer solution was found, the values are immediately updated).

The headings are:

1. An icon showing the type of the variable
2. Column: The column number in the matrix for the current variable
3. Name: The name of the variable, as understood by Mosel
4. Rows: The number of rows(constraints) in which the variable participates
5. LB: The lower bound for the variable
6. UB: The upper bound for the variable
7. Solution: The current solution value of the variable. Check problem status to see if solution is valid!
8. Reduced cost: Current reduced cost. See above for validity.
9. Type: Description of the type of the variable (continuous, binary, integer, semi-continuous, semi-continuous integer, partial integer)

3.9.3 The Row View



	Row	Name	Cols	RHS	Activity
		OBJECTIVE	256		64
≤	0	_R1	24	23	6
≤	1	_R2	40	39	9
≤	2	_R3	2	1	0
≤	3	_R4	2	1	0
≤	4	_R5	2	1	0
≤	5	_R6	10	9	0
≤	6	_R7	6	5	0
≤	7	_R8	12	11	2
≤	8	_R9	30	29	10
≤	9	_R10	12	11	1
≤	10	_R11	52	51	10
≤	11	_R12	8	7	1
≤	12	_R13	56	55	10
≤	13	_R14	8	7	4
≤	14	_R15	4	3	1
≤	15	_R16	6	5	0
≤	16	_R17	2	1	0
≤	17	_R18	36	35	9

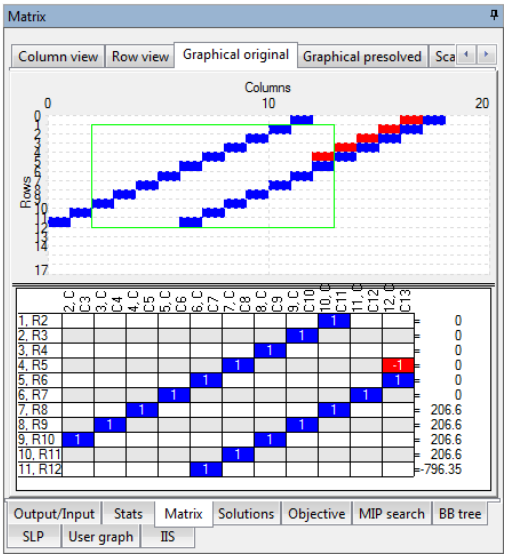
This view lists all constraints (rows) in the matrix currently loaded in the Optimizer.
Note:

- Not all constraints declared in the model may have been sent to the Optimizer: only constraints linked to the Objective function are sent to the Optimizer.
- Values are updated as they become available (*e.g.*, activity values are updated when finding solutions).

The headings are:

1. An icon showing the type of the constraint
2. Row: The row number in the matrix for the current constraint
3. Name: The name of the constraint, as understood by Mosel
4. Cols: The number of variables(columns) with nonzero coefficients in this constraint(row)
5. RHS: The right hand side of this constraint
6. Activity: Its current activity value
7. Slack: Its current slack value
8. Dual: Its current dual value
9. Type: Description of the type of the constraint (=, =,=, range, free)

3.9.4 The Graphical View

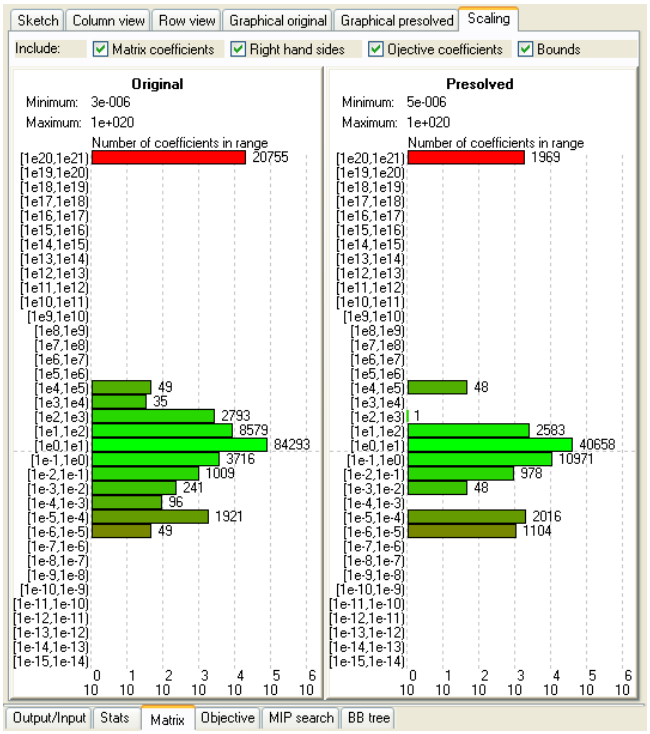


- Use the slider on the left to change the relative size of the two windows.
- If the matrix is small (less than a hundred rows and/or columns), each colored rectangle in the upper part of the window will represent one nonzero coefficient in the matrix.
- If the matrix is large or very large (above one thousand and up to hundreds of thousands of rows and/or columns), a colored rectangle will represent a region in the matrix, corresponding to a certain number of rows and columns. All the nonzero coefficients in the matrix (regardless of its size) are examined; only those regions that have at least one nonzero coefficient will be represented by a colored rectangle.
- The matrix can be zoomed into by clicking and holding the left mouse button and dragging right and down. After zooming, only the region marked by the zooming rectangle will be shown. The coefficients are recounted and the accuracy of the display will increase. After zooming deep in the matrix, a point can be reached where the colored rectangles represent actual coefficients in the matrix. When the current view is the result of a zoom one can move around the matrix by dragging with the right mouse button.
- To zoom out, double-click the left mouse button.
- The rectangle shown permanently next to the cursor acts as a magnifying glass. It marks a region consisting of 12 rows and 10 columns next to the cursor and brings that region into view in the lower part of the window. As the mouse cursor moves around in the matrix, the magnified region is constanly updated.
- The meaning of colors:
  - **Red** — A negative coefficient or a region where all coefficients are negative.
  - **Blue** — A positive coefficient or a region where all coefficients are positive.
  - **Purple** — A region where some coefficients are positive and some coefficients are negative. The shade of purple indicates the predominance of either.
- In the magnified 12 × 10 region, the names of the rows (constraints) and columns (variables) are shown (to enable the actual names used in the Mosel model, set the "mmxprs" boolean parameter XPRS_LOADNAMES to true).

  At the top, following a comma after each column number is the type of variable:

C — indicates a continuous variable;
I — indicates an integer variables;
B — indicates a binary variable;
S — indicates a semi-continuous variable;
R — indicates a semi-continuous integer variable;
P — indicates a partial integer variable.
On the right hand side, the type of constraint is shown. Note that <and >are used instead of the actual $\leq$ or $\geq$, to improve readability. In the case of a non-binding constraint, the character '*' is shown.

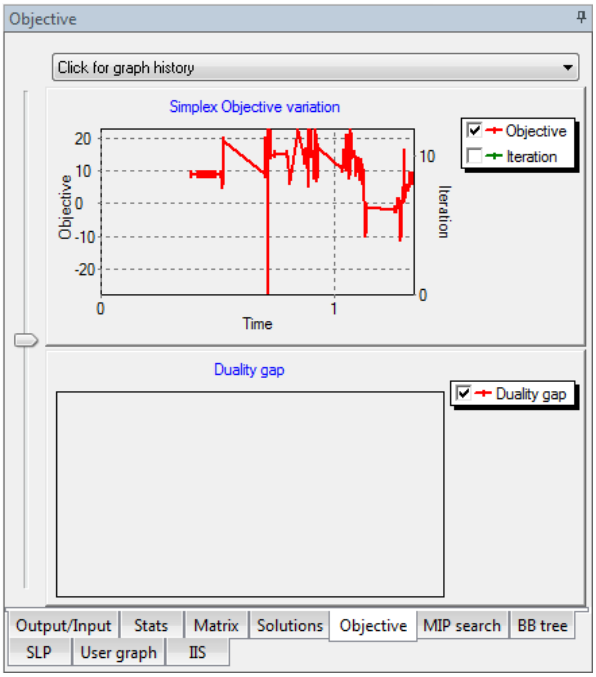
3.9.5 The Scaling View



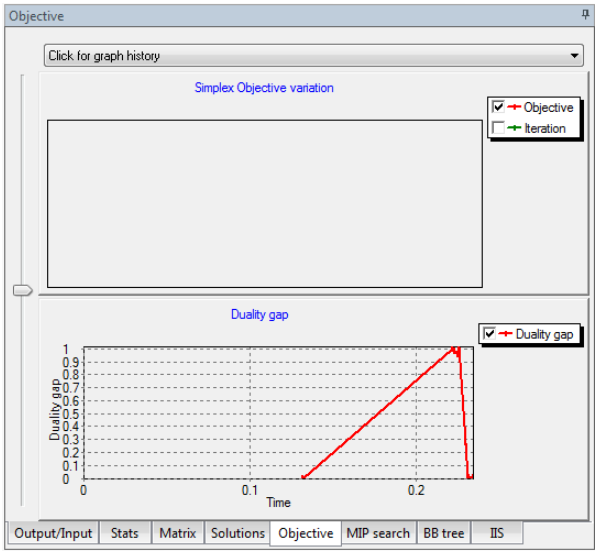
A histogram showing ranges for matrix coefficients, right hand sides, objective coefficients, and bounds.

3.10 The Objective tab of the Run Bar

The graph on top shows the evolution in time of the objective value during the Simplex algorithm. The iteration is also plotted by time.

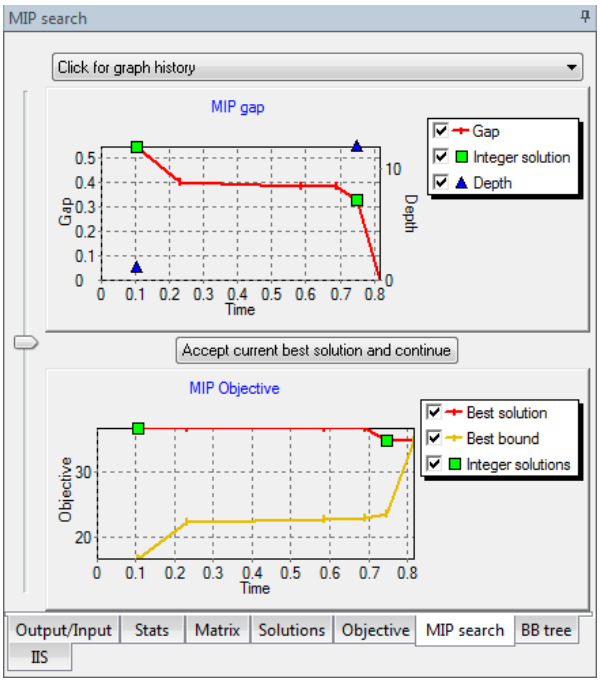


The graph at the bottom shows the evolution in time of the duality gap during the Newton barrier algorithm.



- At any time, only one of these progress graphs can be active (corresponding to the algorithm currently running).
- The graphs can be zoomed in and out and each curve can be shown or hidden using the checkboxes in the graph legend.
- Use the slider bar to change the vertical size of either graph.
- Xpress-IVE maintains a history of the last ten graphs produced. Select the desired graph from the list at the top. Note that the graph history cannot be accessed during an optimization.

3.11 The MIP search tab of the Run Bar

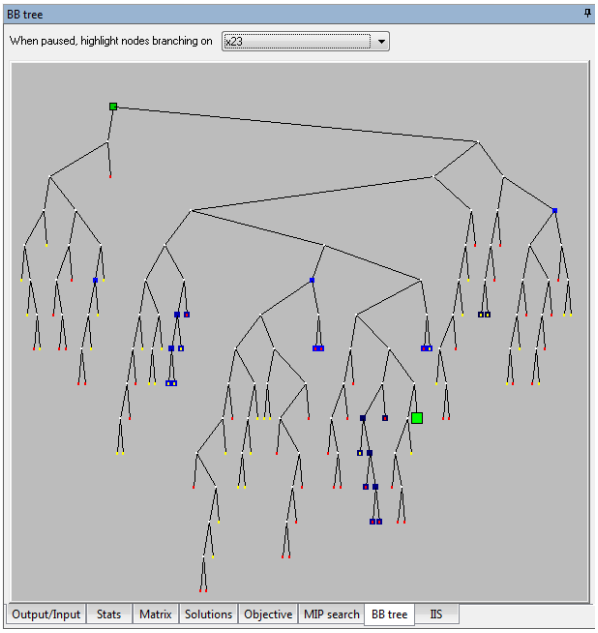


The graph on top shows the evolution in time of the MIP gap during the global search. Also shown are the points where the integer solutions were found and the depth in the search tree at which each solution was found.

The graph below shows the progress of the current best integer solution objective relative to the best bound. Integer solutions are marked as well.

- The graphs can be zoomed in and out and each curve can be shown or hidden using the checkboxes in the graph legend.
- Use the slider bar to change the vertical size of either graph.
- Pressing the button "Accept current best solution and continue" will terminate the MIP search but the execution of the Mosel model will continue. This behavior is unlike that of the *Stop* button, which stops the entire model.
- Xpress-IVE maintains a history of the last ten graphs produced. Select the desired graph from the list at the top. Note that the graph history cannot be accessed during an optimization.

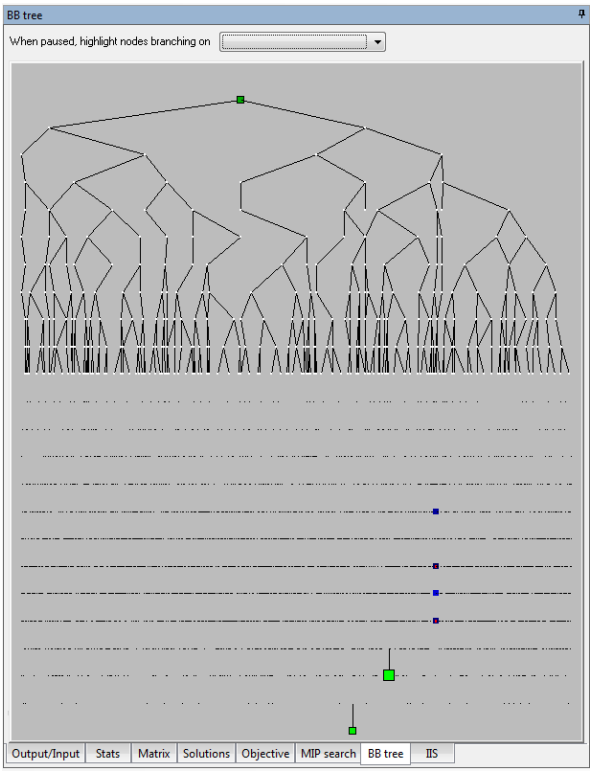
3.12 The BB tree tab of the Run Bar



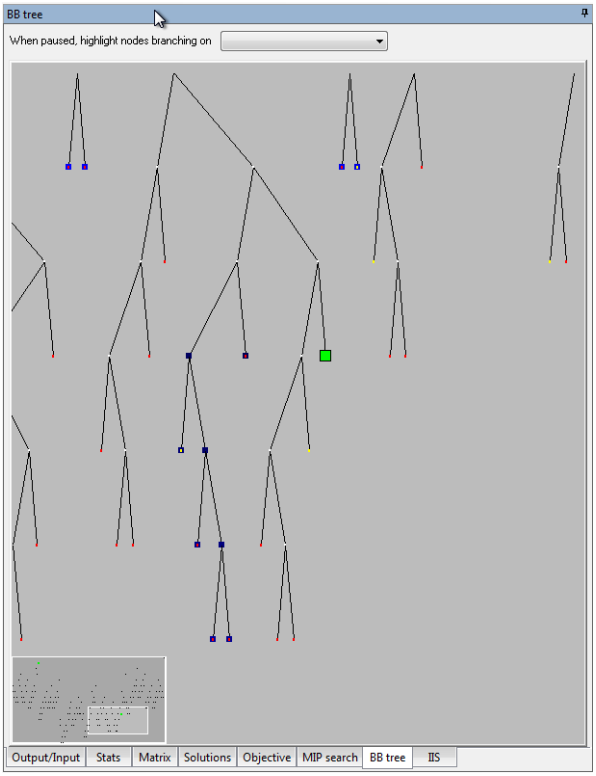
The Branch and Bound tree is a representation of the search for an integer solution when optimizing MIP problems. To enable/disable drawing the BB tree use the *Run Options* dialog.

Notes:

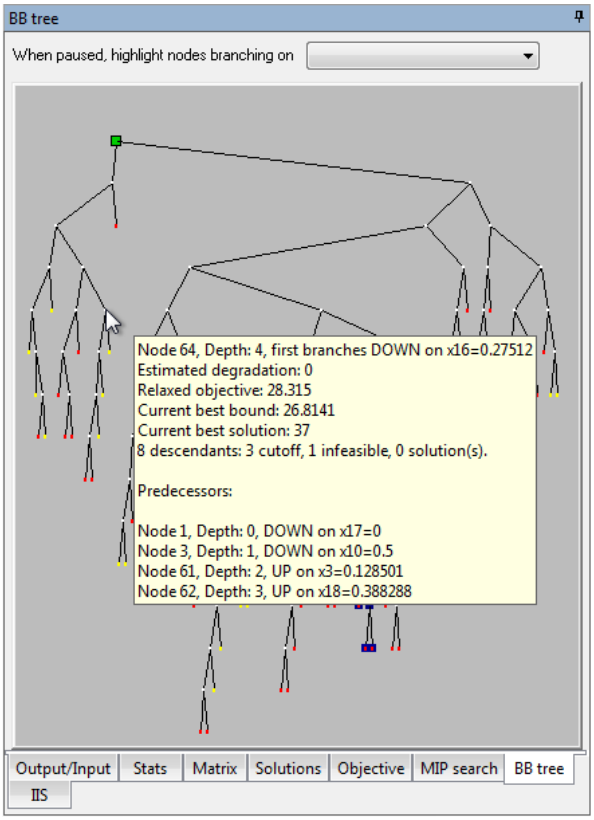
- The tree is displayed for Mosel as well as Optimizer runs and it is updated continuously during the MIP search (to enable the names used in the Mosel model to appear in the tree, set the "mmxprs" boolean parameter XPRS_LOADNAMES to true).
- White nodes are either still active or in-tree.
- Red nodes are infeasible.
- Yellow nodes are cut off.
- The 20 most recent nodes are in decaying shades of blue, with bright blue being the most recent.
- The green nodes represent solutions. The best solution so far is green and large. The brighter the color, the more recent the solution.
- When hovering the mouse above a green solution node, a number will appear next to all solution nodes, indicating the order in which the solutions were found.
- If fewer than 5000 nodes have been visited so far, the entire tree will be drawn (see above).
- If more than 5000 nodes have been visited, only nodes up to a depth of 10 will be drawn fully. The rest of the tree is summarized by scattered dots (which will turn into real nodes when zooming). Solutions and the 20 most recent nodes (in blue) are displayed regardless of their depth:



- Zoom in by holding the left mouse button down, dragging right and down and releasing it. A rectangle will indicate the area to be zoomed.
- Return to full tree view by clicking the left mouse button when zoomed.
- When zoomed, the lower left corner shows a scheme of the entire tree, while the "porthole" (representing the area seen in the main tree window) is highlighted:



- To move around in the tree when zoomed, hold the right mouse button down and drag in any direction. The small preview window in the corner will continuously reflect the new position in the tree.
- The arrow keys will also move around in the tree (ensure the tree window has the focus by right-clicking in it - do not left click – that will return to full tree view).
- If lost in the tree or if the image doesn't look correct, left click to return to full view.
- Hovering the mouse above nodes will produce tooltips with information on the current node.
- The tooltips at each non-terminal node contain the following information:
  – Node number
  – Depth
  – Initial branching direction
  – The branching variable and its current relaxed solution value
  – Relaxed objective
  – Current best bound
  – Current best solution
  – Number of descendants
  – How many descendants are cutoff, infeasible, solutions
  – The ten most recent ancestors of the node, each with info on branching direction



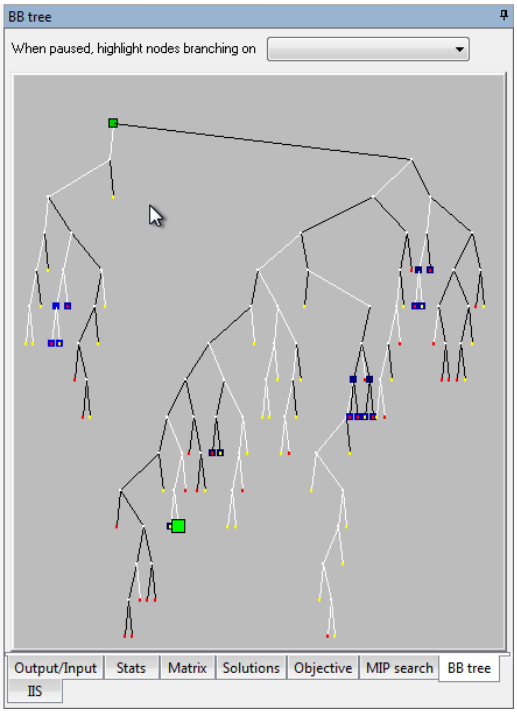
- Double click on a node to obtain a listing of all of its ancestors (shown in the *View* text dialog).
- When the optimization is complete, paused or stopped, search for specific branching variables in the tree using the tree highlighter tool (can be enabled/disabled from the *Run Options* dialog): just select a variable name from the drop list and all visible nodes branching on that variable will begin to flash in alternating blue and yellow.

- To stop highlighting, select the first item in the highlighter tool droplist: the blank.
- While there is no limit on the size of the displayed tree, if the tree is very large (e.g. over 200,000 nodes on a PIII 800Mhz machine), it will react slowly to user actions.

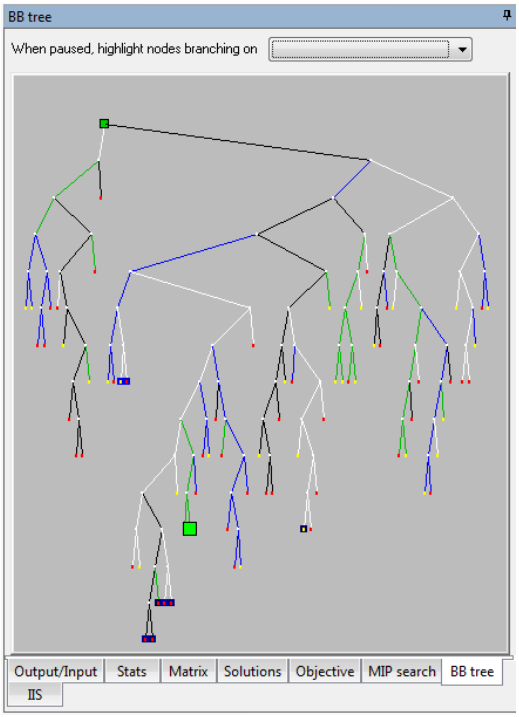
3.12.1 Parallel branch and bound trees

Note that each branch is colored according to the thread that processed the node. Shown below are trees obtained with 2, 4, and 8 threads.

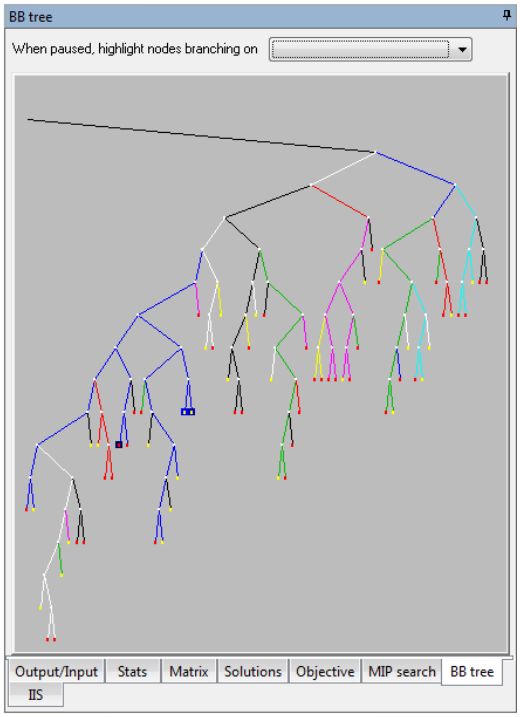
2 threads



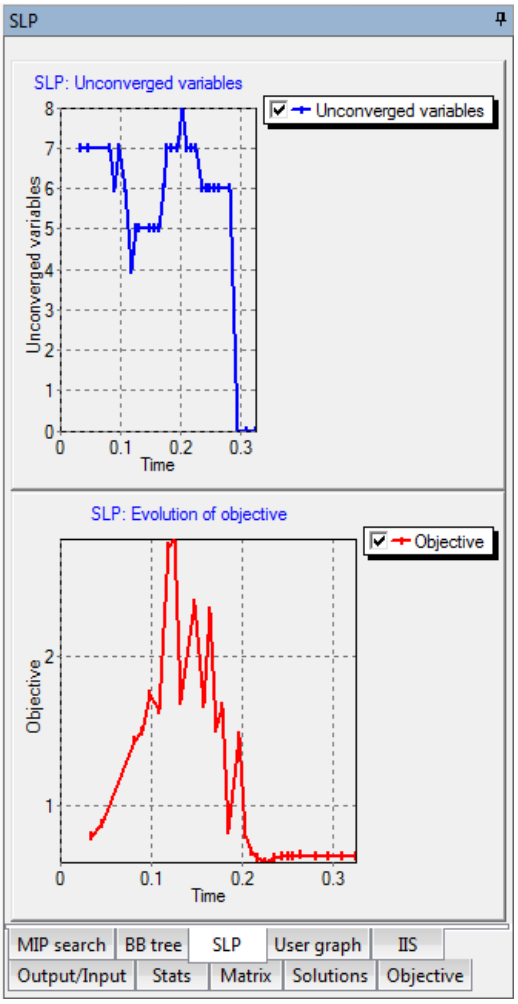
4 threads



8 threads



3.13 The SLP search tab of the Run Bar

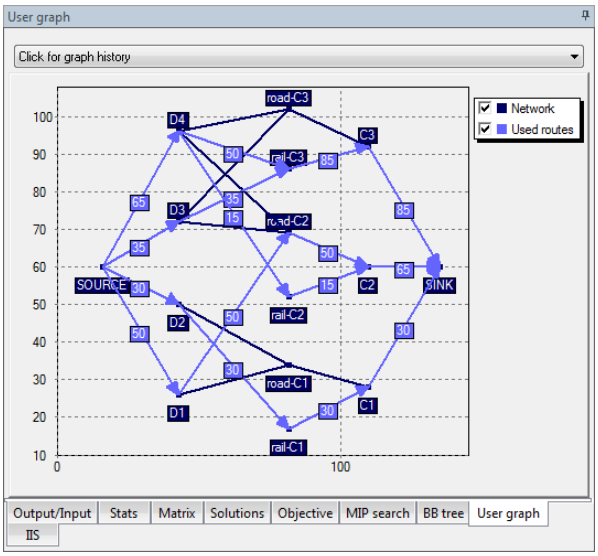


The graph at the top shows the evolution in time of the number of unconverged variables during the SLP run. A sample is taken at each SLP iteration.

The graph at the bottom shows the progress of the current best objective. A sample is taken at each SLP iteration.

- The graphs can be zoomed in and out and each curve can be shown or hidden using the checkboxes in the graph legend.

3.14 The User graph tab of the Run Bar



This window offers the opportunity to plot points on an unlimited two-dimensional grid when using the "mmive" Mosel library. The graph is automatically scaled to include all and only the plotted points. There is no limit on the number of items that can be plotted.

The "mmive" module in Mosel contains these functions and procedures:

`procedure IVEpause( message: string)` Pauses the Mosel execution at the line where it was called. It also prints a message at the top of the Run Bar that may inform the user of the reason for pausing. While the execution is paused, model entities can be examined in the entity tree, or the user graph may be inspected in slow motion.

`function IVE_RGB( red: integer, green: integer, blue: integer): integer` Compute a composite color by combining amounts of red, green and blue.

`function IVEaddplot(name:string, color:integer): integer` Inserts a new plot on the user graph. A plot is indentified by its name and can be shown or hidden using its corresponding legend checkbox. The maximum number of distinct plots is currently limited to 20. However, each plot can contain an unlimited number of points, lines, arrows and labels. In the graph above, both "first fractal" and "second fractal" are plots. They can be shown/hidden using the checkboxes in the legend.

`procedure IVEdrawarrow(handle:integer, x1:real, y1:real, x2:real, y2:real)` Add an arrow to an existing plot. The arrow connects the two points whose coordinates are given as parameters, pointing to the second one.

`procedure IVEdrawlabel(handle:integer, x:real, y:real, text:string)` Add a text box to an existing plot. The box will be centered horizontally just above the point given.

`procedure IVEdrawline(handle:integer, x1:real, y1:real, x2:real, y2:real)` Add a line to an existing plot. The line connects the two points whose coordinates are given as parameters.

`procedure IVEdrawpoint(handle:integer, x:real, y:real)` Add a small square to mark a point at the given coordinates.

`procedure IVEerase` Remove all plots and reset the user graph.

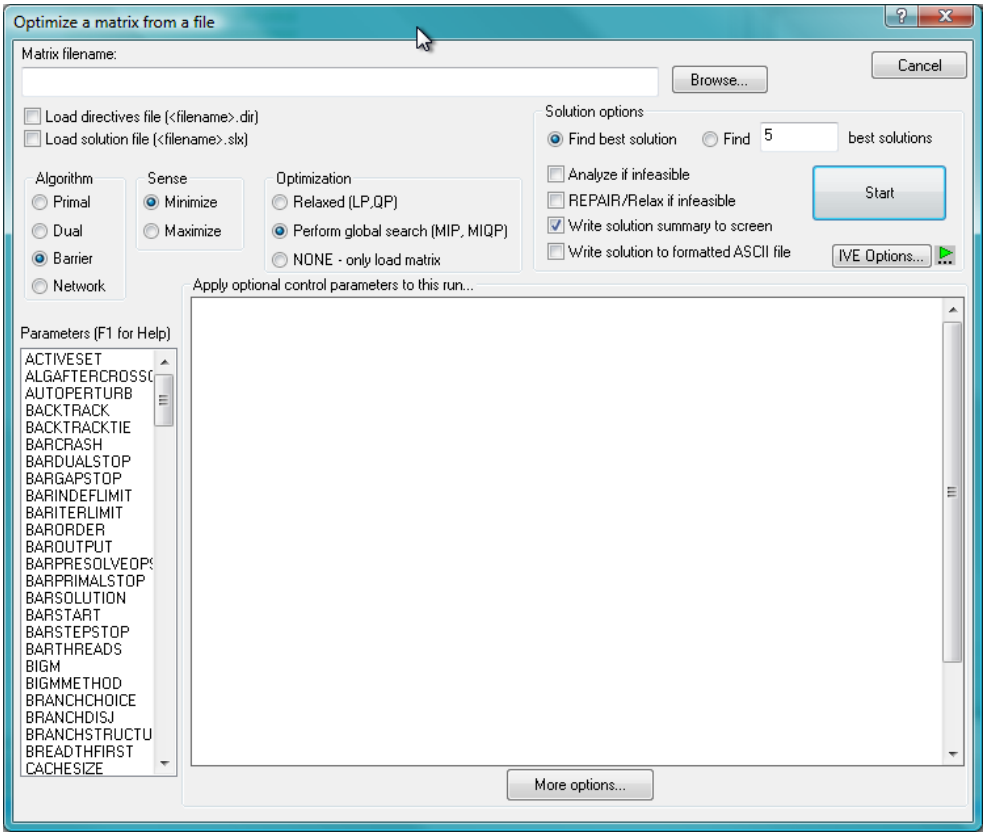
`procedure IVEzoom(x1:real, y1:real, x2:real, y2:real)` Scales the user graph. The viewable area is determined by its lower left and upper right corners.

CHAPTER 4

Xpress-IVE Dialogs

4.1 Optimizer Dialog

Accessible by selecting *Optimize matrix file...* from the *Optimizer* menu or by clicking the / / buttons on the Toolbar.



The *Optimizer* dialog allows the optimization of a matrix file (MPS or LP format). After the various settings have been set, the *Start* button will proceed with the optimization. Up to five different strategies may be 'checked'. IVE will run the strategies successively and display the progress log in the *Output* window. To compare the relative performance of different control parameter settings visually, look at the histories of the MIP search graphs.

Options:

4.1.1 Load directives file

Perform the branch and bound search according to an Optimizer 'directives' file. Please check the Optimizer reference manual for more information on directives and how to set them.

4.1.2 Load solution file

Loads an .slx solution file prior to solving an MIP problem. Please check the Optimizer reference manual for more information on .slx solution files.

4.1.3 Algorithm

Primal	Simplex primal.
Dual	Simplex dual.
Barrier	Newton barrier interior point.
Network	Network.

4.1.4 Sense

Note that this setting will override the sense specified in the LP file. Make sure the correct sense is selected.

Minimize	Minimize the objective function
Maximize	Maximize the objective function

4.1.5 Optimization

Relaxed	Find a relaxed solution
Perform global search	Find integer solutions
NONE – only load matrix	Loads the matrix for visualization/inspection. No optimization is performed.

4.1.6 Solution options

Find best solution	For MIP problems: runs the Optimizer to find the optimal solution.
Find N best solutions	For MIP problems: runs the Optimizer in a special mode to find alternate optimal solutions. (Note that the BB tree visualization will no longer be accurate)
Analyze if infeasible	If problem is infeasible, run the IIS Optimizer command to examine infeasibility sets.
REPAIR/Relax if infeasible	Attemps to find a feasible solution to an infeasible problem by relaxing constraints and bounds. For more information look up the command REPAIRINFEAS in the Optimizer Reference Manual.

Write solution summary to screen Write nonzero variable values from the solution vector to the Output window in the Run Bar.

Write solution to formatted ASCII file Write the solution to a .prt file and open it for viewing.

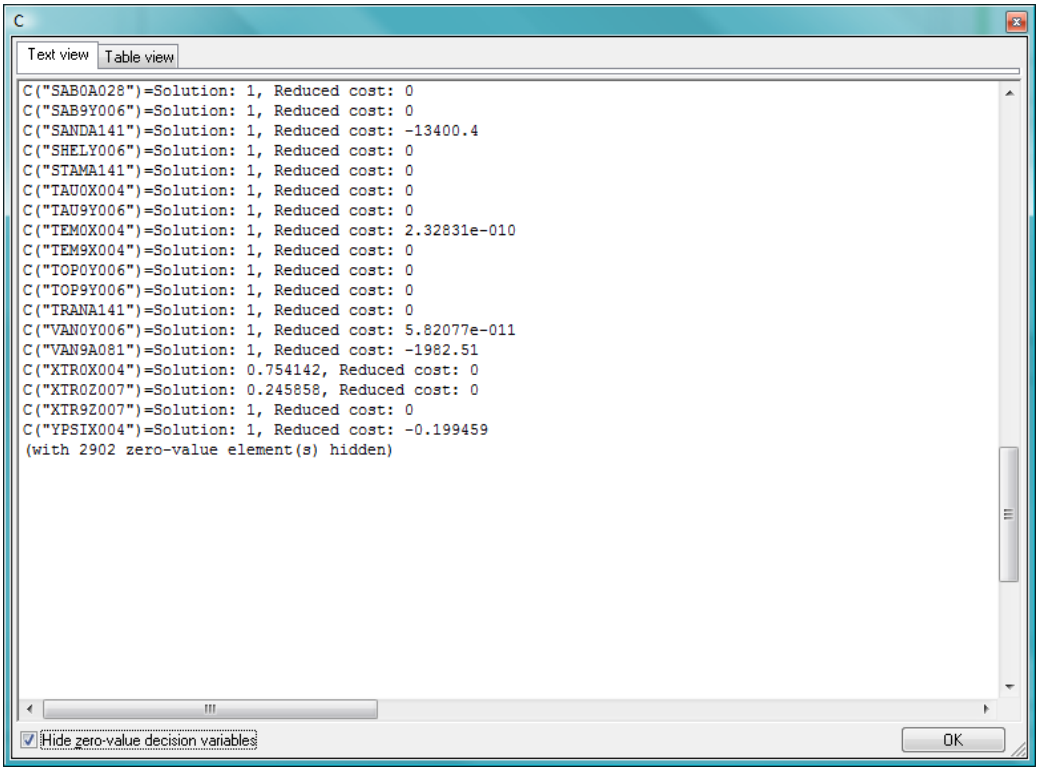
4.1.7 Strategies (control parameter settings)

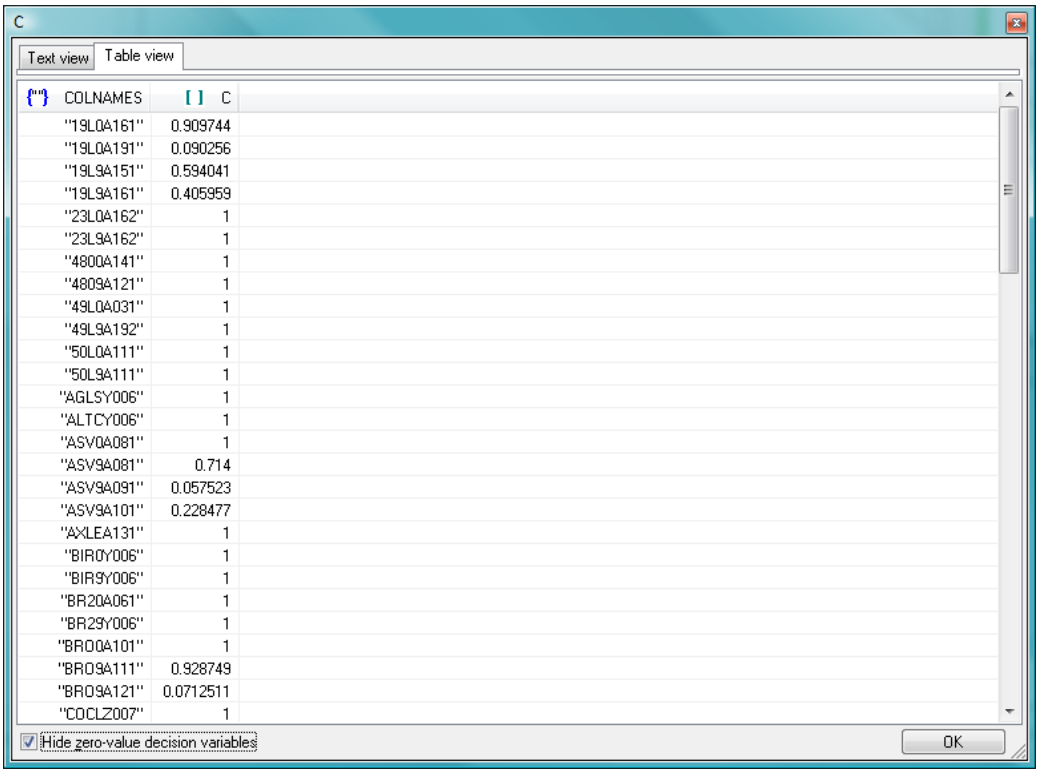
Set these control parameters before the matrix is read in.

For example:

```
VARSELECTION=3 MIPRELSTOP=0.05
```

4.2 View text Dialog





The *View* text dialog can be used to display in simple editable format:

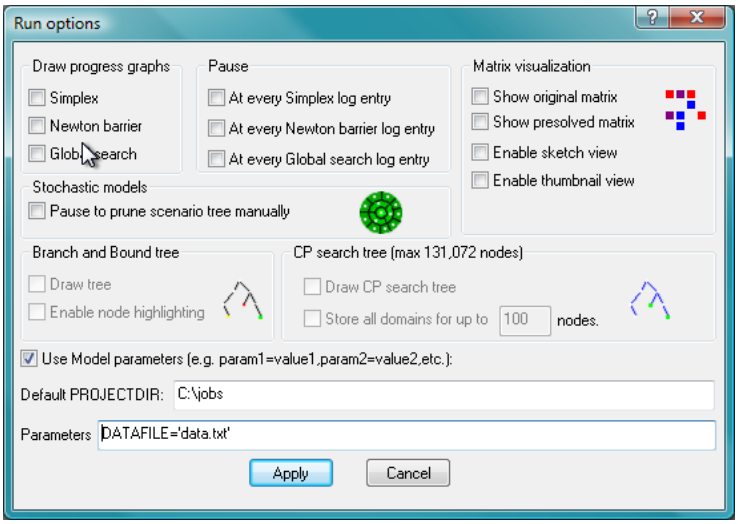
- The value of an entity in the Entity tree, by double clicking the entity. This is where the full listing of an array or set can be examined.
- The value of an entity in the editor window, by bringing up the context menu and selecting *Show value of...*. This only applies to non-scalar entities, such as arrays and sets.
- The complete list of ancestors for a node in the branch and bound tree pane in the Run Bar, after double clicking the node.

The contents of either the text view or the table view can be saved (as plain text or as CSV respectively) or copied to the clipboard via the context menu on each control. The text menu further offers facility for full cut-and-paste, and text search.

There is one control to affect display of data: The *Hide zero-value decision variables* checkbox will, when checked, remove those elements from a collection of mpvar for which the Mosel `getsol` function returns zero.

4.3 Run options Dialog

Accessible by selecting *Options...* from the *Build* menu or by clicking the  button on the Toolbar.



Settings that control the execution of a Mosel model or the optimization of a matrix file can be modified using this dialog. This dialog can be invoked at any time, even while a run is taking place. Changes will take effect immediately after the dialog is dismissed.

Options:

4.3.1 Draw progress graphs

Simplex Graph the progress of the Simplex algorithm. The sampling is made every second.

Newton barrier Graph the progress of the Newton barrier algorithm.

Global search Graph the progress of the global search for integer solutions. The sampling is made every second.

4.3.2 Branch and bound tree

Draw tree Draw the Branch and Bound tree.

Enable node highlighting Allow searching for nodes in the tree by variable name.

4.3.3 Pause

At every Simplex log entry Pause at every log entry during the Simplex algorithm. Log entries are produced periodically during the course of the Simplex algorithm; their frequency is controlled by the "mmxprs" parameter XPRS_LPLOG.

At every Newton barrier log entry Pause at every iteration during the Newton barrier algorithm.

At every Global search log entry Pause at every log entry during the global search for integer solutions. The frequency with which log entries are generated during the global search is controlled by the "mmxprs" parameter XPRS_MIPLOG.

4.3.4 Matrix Visualization

Show original matrix After a problem is loaded in the Optimizer, its matrix can be examined. Select this option to make a copy of the matrix and visualize it.

Show presolved matrix After a problem is 'presolved' by the Optimizer, the matrix will most likely have changed. Select this option to examine the presolved matrix.

4.3.5 Stochastic models

(if available)

Pause to prune scenario tree manually When developing a stochastic model, use this option to pause the Mosel run just before solving the stochastic problem. When the model is paused, scenarios can be aggregated or deleted in the scenario tree.

4.3.6 Constraint programming models

(if available)

Draw CP search tree Draw the CP search tree.

Store all domains for up to nnnn nodes Maintains detailed variable domain information for the first nnnn nodes.

4.3.7 Use Model parameters

If the Mosel model has declared one or more parameters, they can be overriden. Select this option and then specify the parameter values to be used during the execution. Any number of parameters may be omitted: only those specified will modify the default parameter values.

4.3.8 Default PROJECTDIR

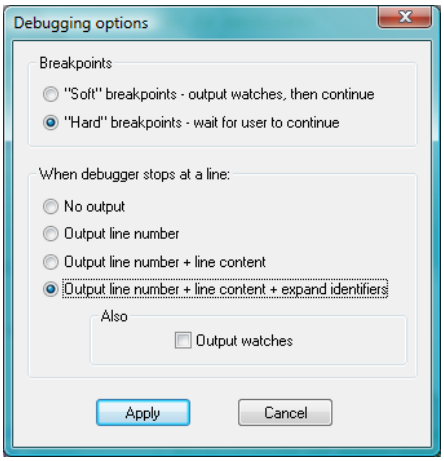
If a Project is loaded, this will be set by default to the directory in which the Project is located.

4.3.9 Parameters

A sequence of assignments, in `name=value, name2=value...` format, specifying names and values to be passed to the `parameters ... end-parameters` block of a Mosel model when it runs.

4.4 Debug options Dialog

Accessible by selecting *Debug Options...* from the *Debug* menu or by clicking the  button on the Toolbar.



This dialog contains settings that control the behavior of the debugger.

Options:

4.4.1 Breakpoints

"Soft" breakpoints	When reaching a soft breakpoint, perform any of the actions below and then continue. This helps in gathering a log of the model execution without user intervention.
"Hard" breakpoints	When reaching a hard breakpoint, watches are updated and the execution stops. The execution must be resumed manually.

4.4.2 When debugger stops

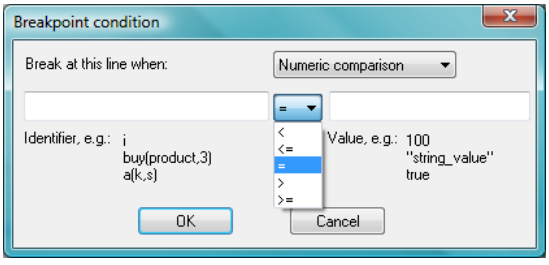
The following options are available:

No Output
Output line number
Output line number + line content
Output line number + line content + expand identifiers
Output watches

See some typical debugger output for more information.

4.5 Breakpoint condition Dialog

Accessible by selecting *Set/Remove Breakpoint condition...* from the *Debug* menu or by clicking the  button on the Toolbar.

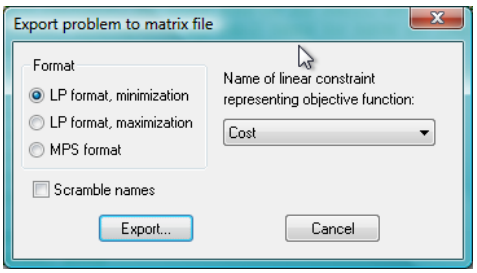


This dialog allows setting a conditional breakpoint.

The conditional breakpoint will activate only when the condition is met.

4.6 Export to matrix Dialog

Accessible by selecting *Export Matrix...* from the *Build* menu or by clicking the  button on the Toolbar.



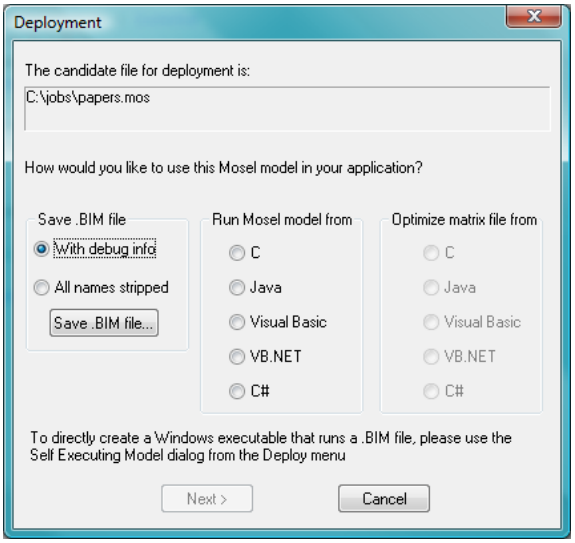
After the Mosel run completes successfully, a standard matrix file (MPS or LP) can be exported using this dialog.

The objective function in the matrix file is determined by the linear constraint entity specified. Consequently, the objective must have been declared as a linear constraint variable in the Mosel model.
The column and row names can be scrambled to hide the original entity names from the model.

4.7 Deploy Dialog

Accessible by selecting *Deploy...* from the *Deploy* menu or by clicking the  button on the Toolbar.

If you wish to build a model as an executable file, see the Self Executing Model Dialog.



This dialog is the first step in deploying a mosel model or a matrix file in a user application written in C/C++, Java or VB. After selecting how the current file will be deployed, click on "Next >" to preview and save the generated code.

Choices:

Save .BIM file (with debug information) The .BIM file will contain all original strings plus debug information.

Save .BIM file (all names stripped) Secure the .BIM file by removing human-readable identifier names.

Run Mosel model from C Only available when a .mos file is open in the editor. Produces a simple C program.

Run Mosel model from Java Only available when a .mos file is open in the editor. Produces a simple Java program.

Run Mosel model from Visual Basic Only available when a .mos file is open in the editor. Produces a simple VB program.

Run Mosel model from VB.NET Only available when a .mos file is open in the editor. Produces a simple VB.NET program.

Run Mosel model from C# Only available when a .mos file is open in the editor. Produces a simple C# program.

Optimize matrix file from C Only available when a matrix file is open in the editor. Produces a simple C program.

Optimize matrix file from Java Only available when a matrix file is open in the editor. Produces a simple Java program.

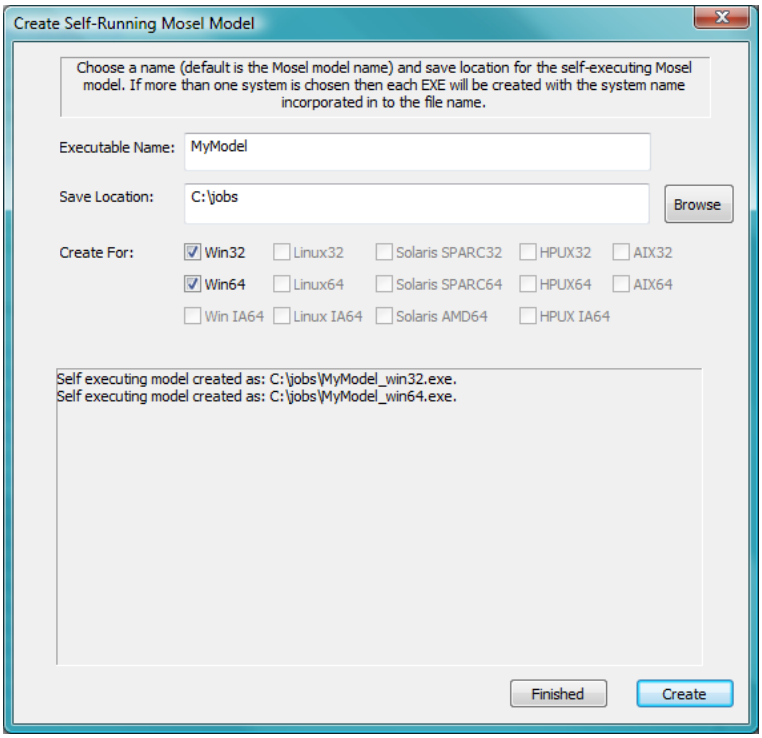
Optimize matrix file from Visual Basic Only available when a matrix file is open in the editor. Produces a simple VB program.

Optimize matrix file from VB.NET Only available when a matrix file is open in the editor. Produces a simple VB.NET program.

Optimize matrix file from C# Only available when a matrix file is open in the editor. Produces a simple C# program.

4.8 Self-Executing Model Dialog

Accessible by selecting *Self-Executing Model...* from the *Deploy* menu or by clicking the  button on the Toolbar.



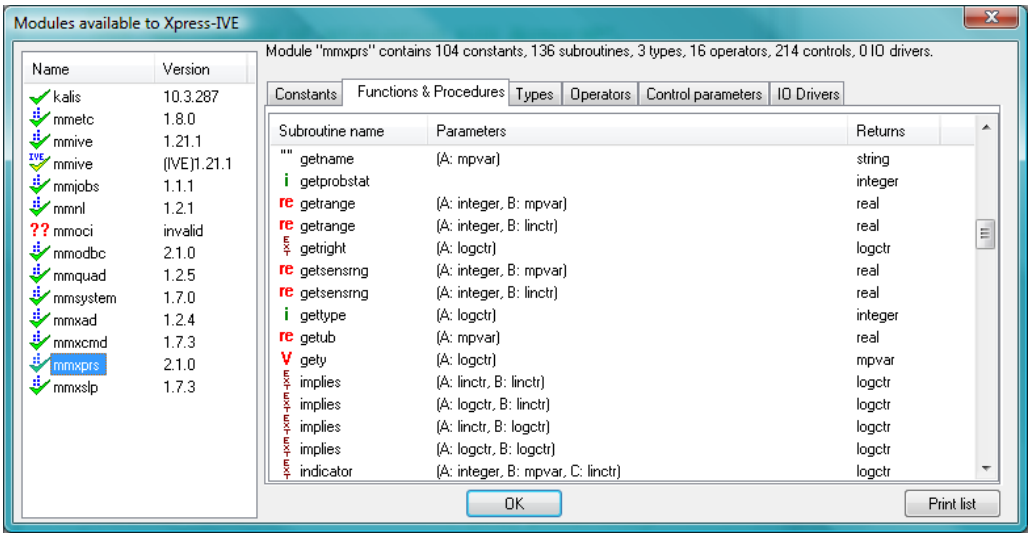
This dialog can be used to create executables with self-contained compiled Mosel models within them.

The dialog contains the following options:

- **Executable Name:** The name of the executable to create. The file used to create the self-running model will be the Mosel file currently in focus in the editor. Executables are created using a concatenation of this name and the platform.
- **Save Location:** The folder location to attempt to save any created executables to.
- **Create For:** Any platforms for which IVE finds suitable executable stubs for will be selectable. New executable stubs will be added over time and should be available from the Xpress website. Currently there are only Windows stubs available. Any stub executables (named SelfRunMosel_platform.exe) should be placed in the "%XPRESSDIR%\bin\Tools_SRAssist" folder.
- **Message Output:** Any error messages or notifications concerning the executable creation are displayed here.
- **Finished:** Exit the dialog.
- **Create:** Click this to have IVE attempt to build and create the executable for the selected platforms.

4.9 List Modules Dialog

Accessible by selecting *List available modules...* from the *Modules* menu or by clicking the  button on the Toolbar.

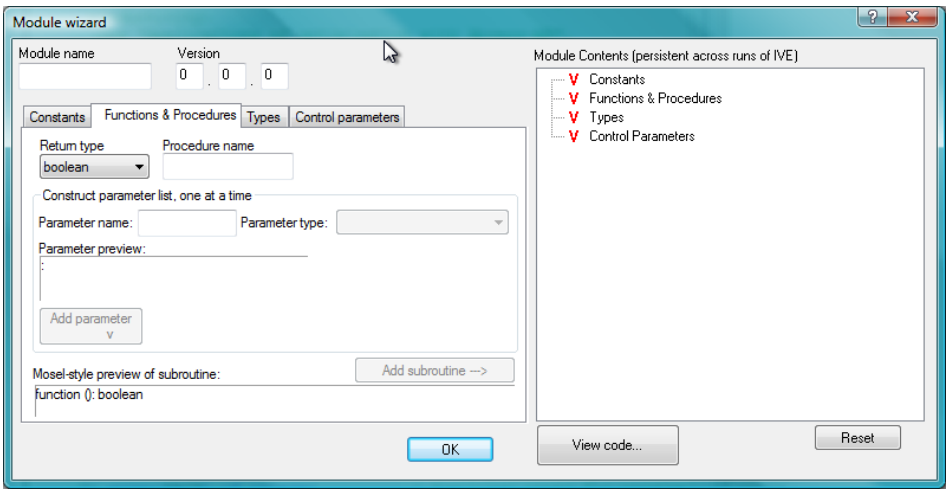


Lists the contents (constants, subroutines, operators, types and control parameters) of Mosel dynamic modules that are currently available to Mosel. Please refer to the Mosel documentation for more information on what modules are.

- Select a module name from the list on the left and then browse through its contents using the tabs on the right.
- A green check next to a module name indicate that it was successfully loaded by Mosel.
- A green check with a Dash Optimization logo indicates a module which is part of the Xpress package.
- Two red question marks next to a module name indicate that even though a .dso file exists, it could not be loaded by Mosel.

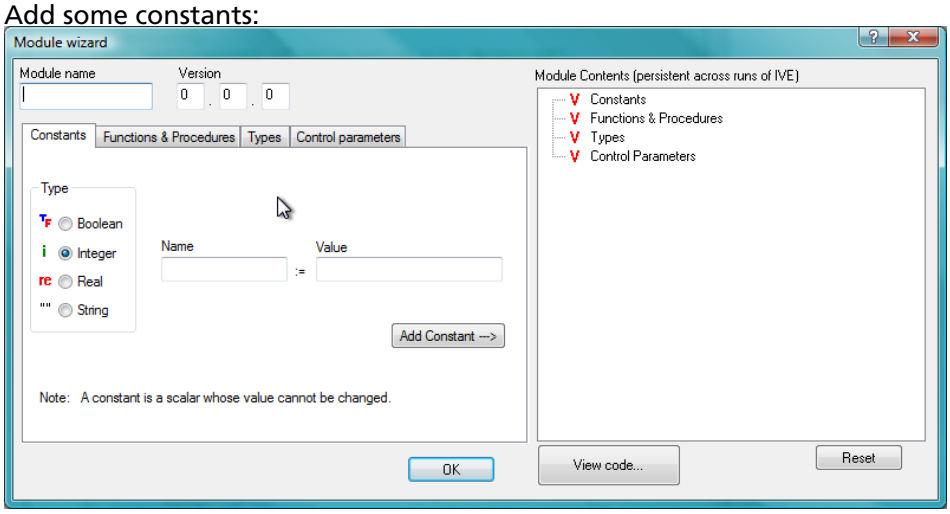
4.10 New Module Wizard Dialog

Accessible by selecting *Create a new module...* from the *Modules* menu or by clicking the  button on the Toolbar.

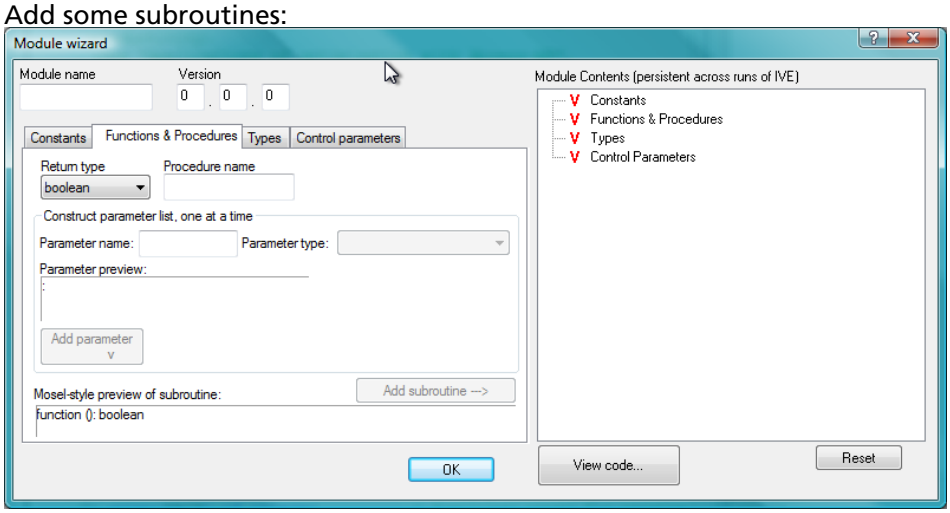


Edit the contents of a virtual .dso module (constants, subroutines, types and control parameters), then generate the template source code for producing the module.

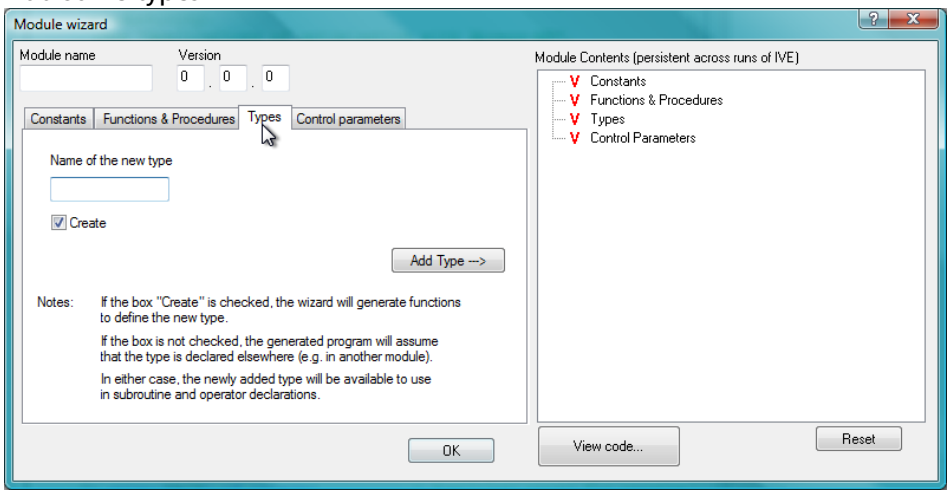
- Name the module and specify version numbers.
- Add some constants:



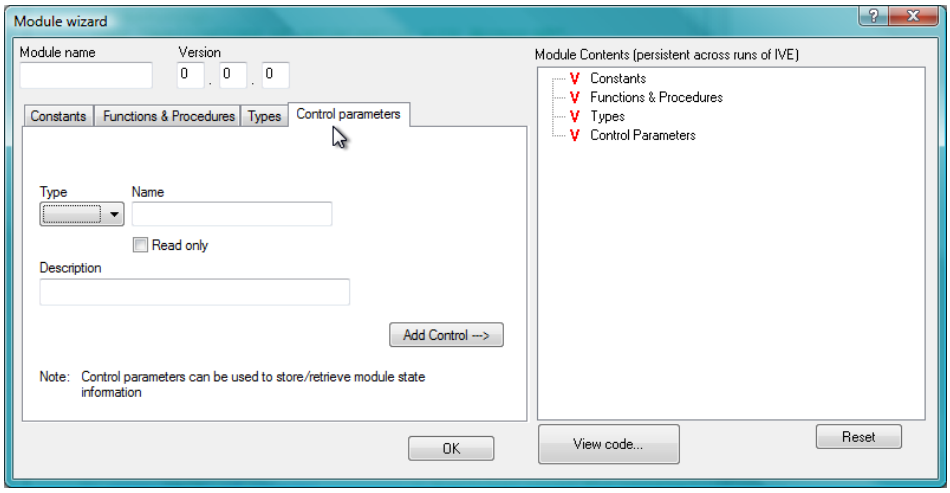
- Add some subroutines:



- Add some types:



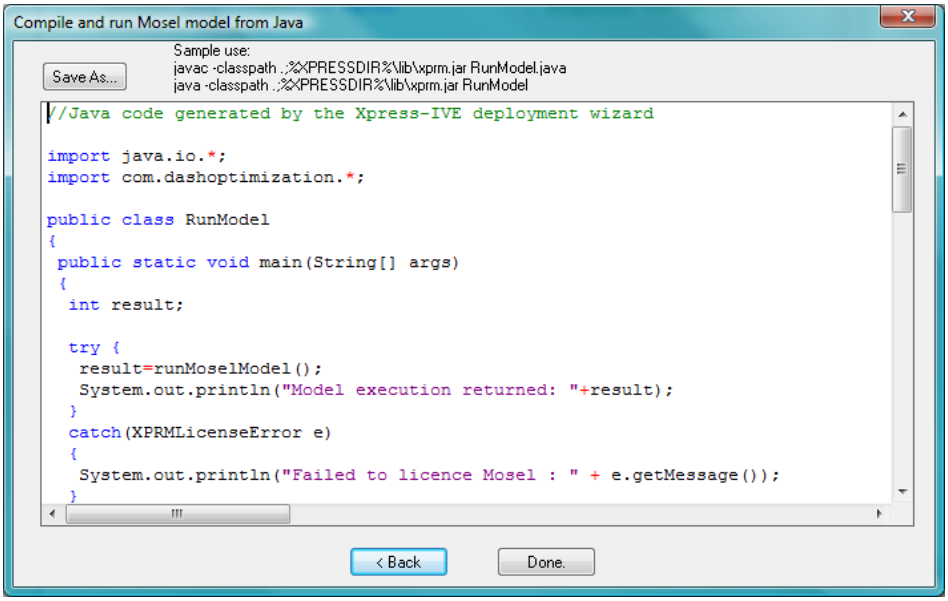
- Add some control parameters:



- At any time, clicking on *View code...* will show the Source code dialog with the code generated for the current contents of the module.
- When satisfied with the contents of the module, save the code that is produced and fill in the functionality.

4.11 Source code Dialog

Shown after completing the steps required by the *Deploy* or the *New Module Wizard* dialogs.



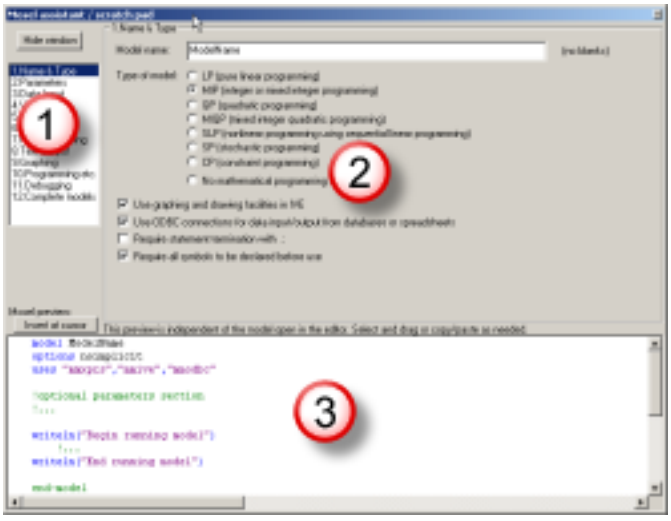
This dialog is the last step in any Xpress-IVE operation that produces C, Java, VB, VB.NET or C# source code. Depending on what type of source code was produced, compilation instructions are given. Save the contents of the editor using the *Save As...* button.

CHAPTER 5

Xpress-IVE Wizards

Wizards can be used to learn more about Mosel, simplify repetitive tasks when developing models and/or provide insight into the more advanced features of the Mosel modeling language and the Optimizer.

The general structure of each wizard window is:

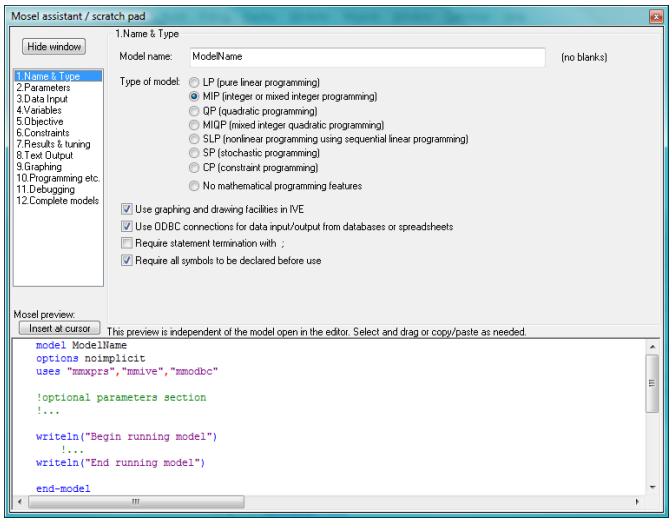


1. **Wizard selector:** at any moment, select the desired wizard from here. Skip any number of steps, focus on only one wizard, *etc.*
2. **Actual wizard:** this part is specific to each of the current 12 wizards.
3. **Mosel source:** source code produced while interacting with the wizard. This code can be edited, inserted in the Mosel editor or copy/pasted.

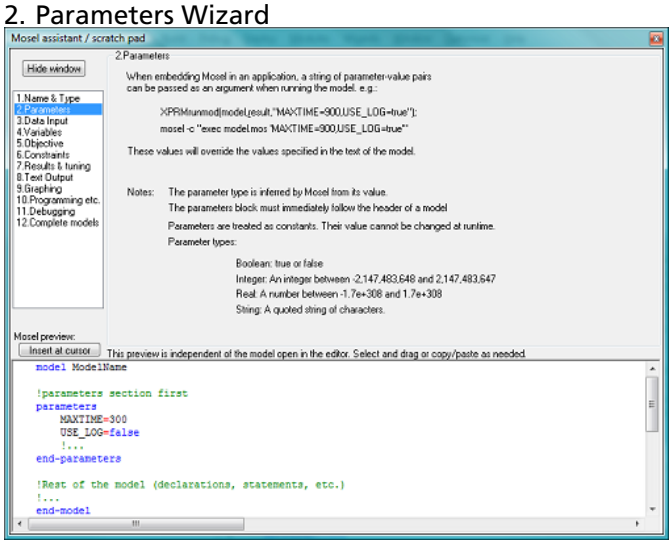
Wizards are designed to be self explanatory and highly interactive. Use the mouse to select options and watch how the Mosel source code is updated for the current task. Interacting with the wizards is the only way to assess the practicality of using a wizard for a task. Selecting options in the wizards is entirely harmless and reversible. No changes are made to your main Mosel model without explicit approval (*e.g.*, pressing the *Insert at cursor* button, which adds the wizard-generated code to the Mosel model opened in the editor).

Select a wizard below for more information:

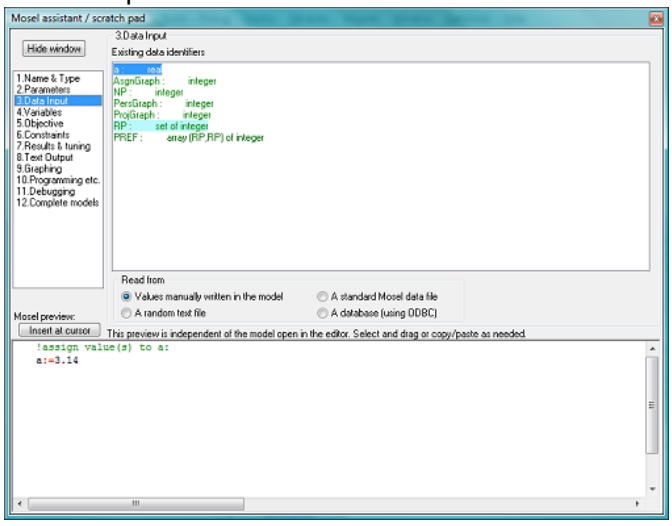
- 1. Name & Type Wizard



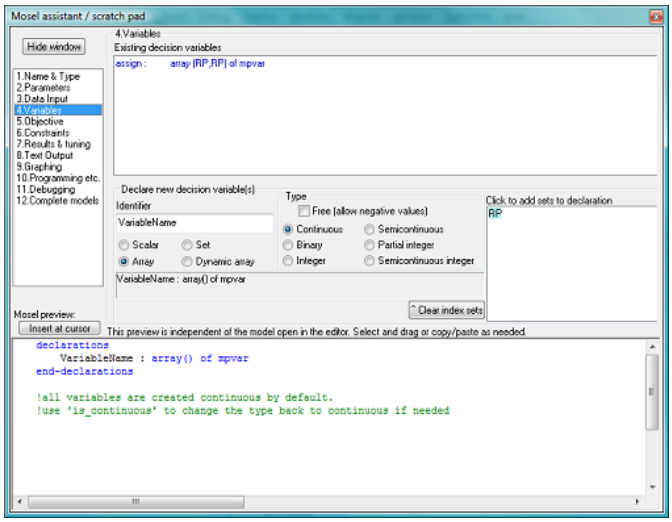
■ 2. Parameters Wizard



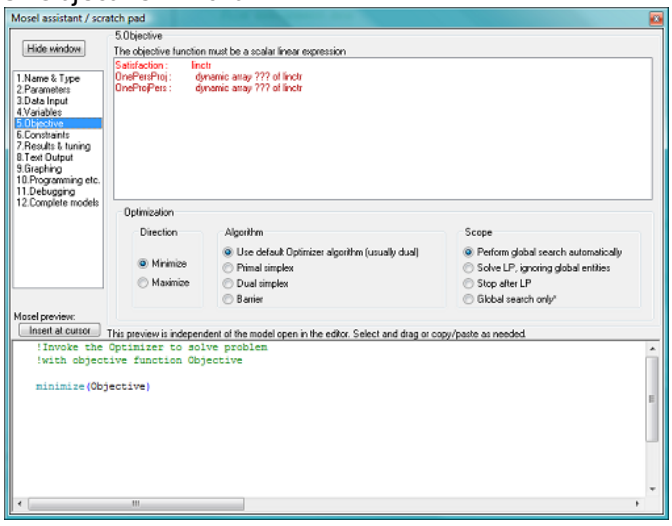
■ 3. Data Input Wizard



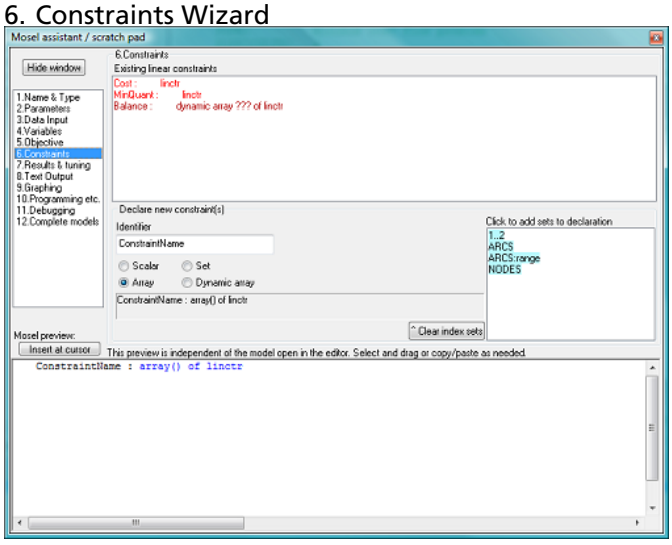
■ 4. Variables Wizard



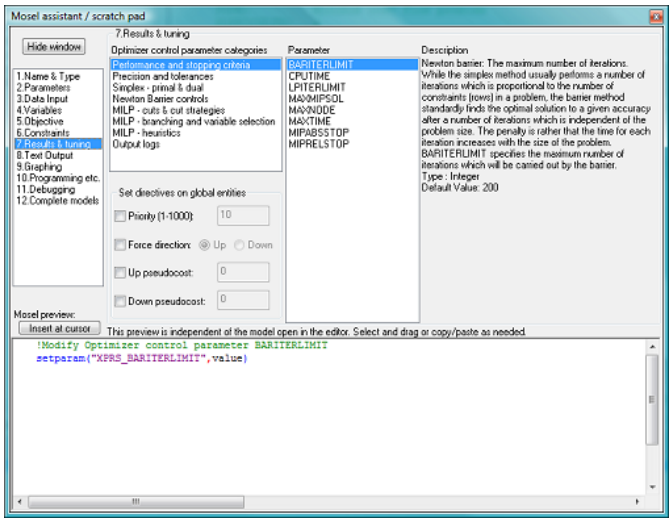
■ 5. Objective Wizard



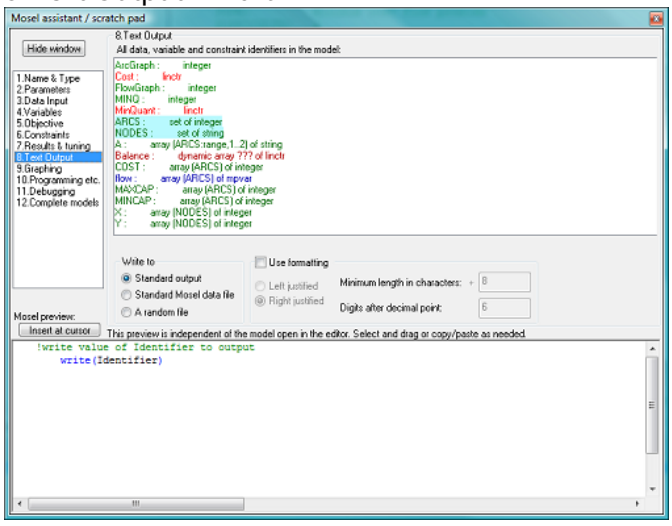
■ 6. Constraints Wizard



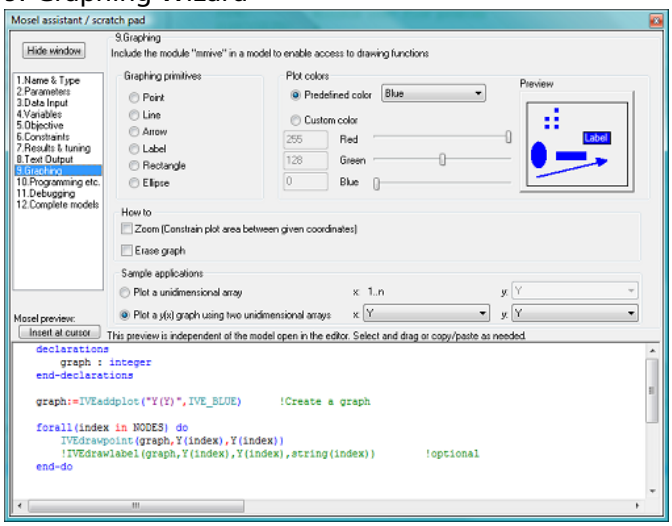
■ 7. Results & tuning Wizard



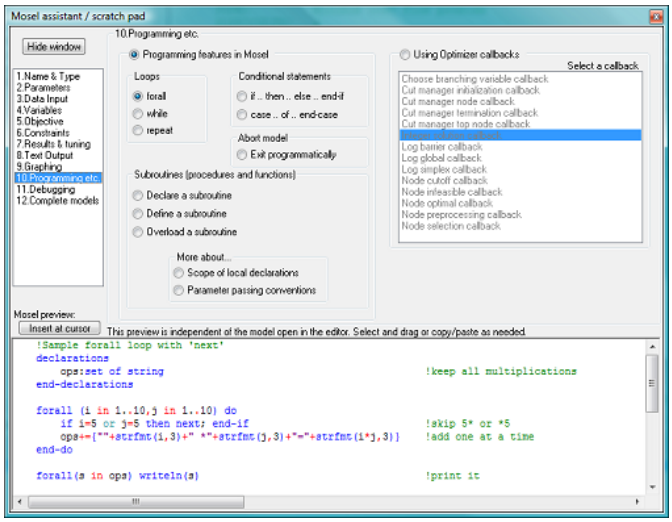
■ 8. Text Output Wizard



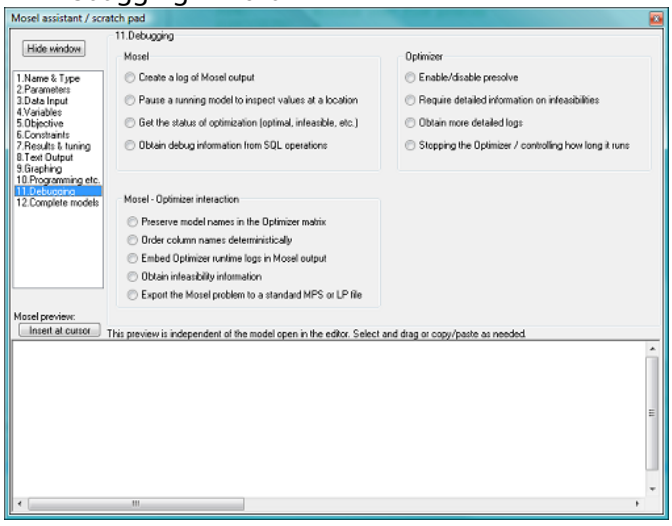
■ 9. Graphing Wizard



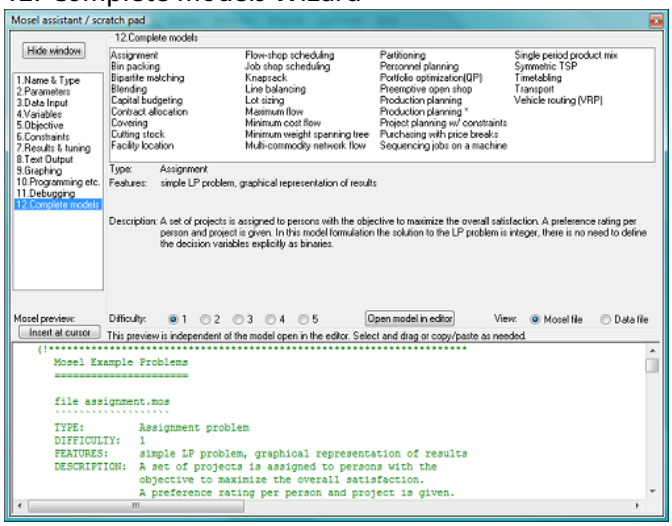
■ 10. Programming Wizard



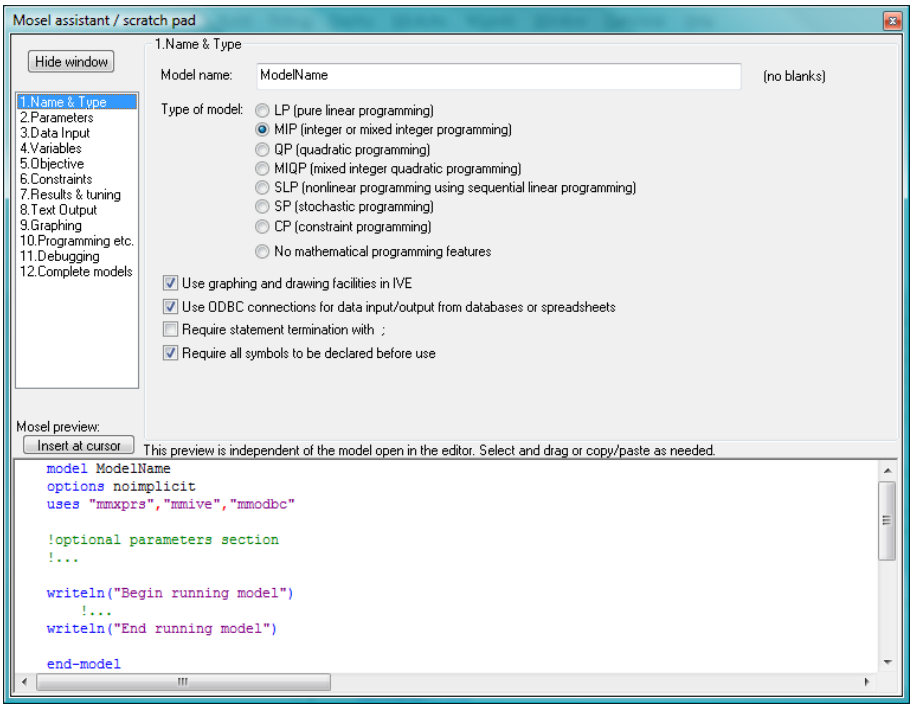
11. Debugging Wizard



12. Complete models Wizard

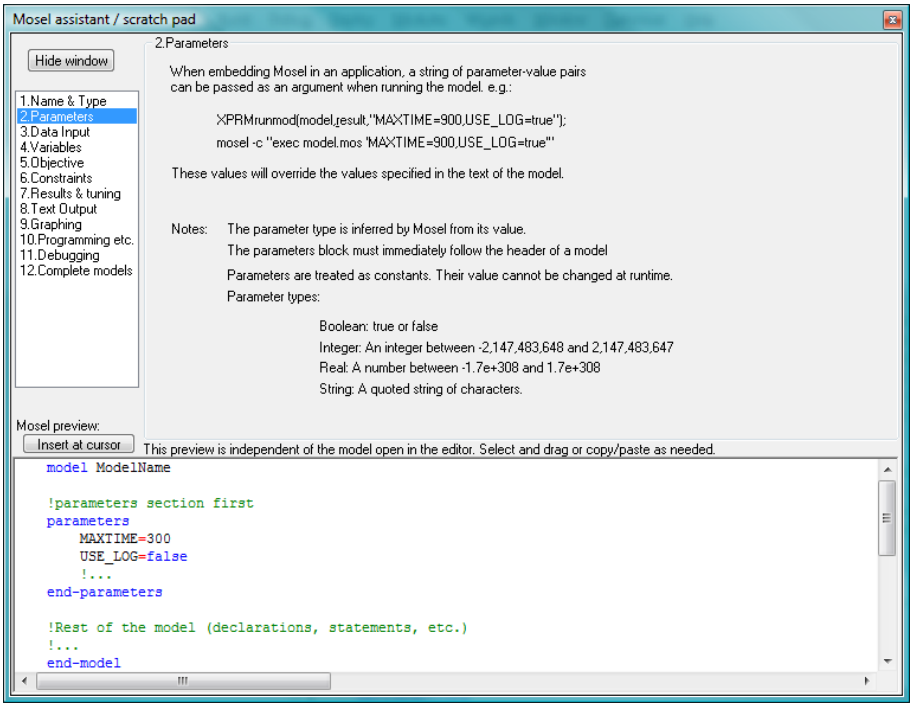


5.1 1. Name & Type Wizard



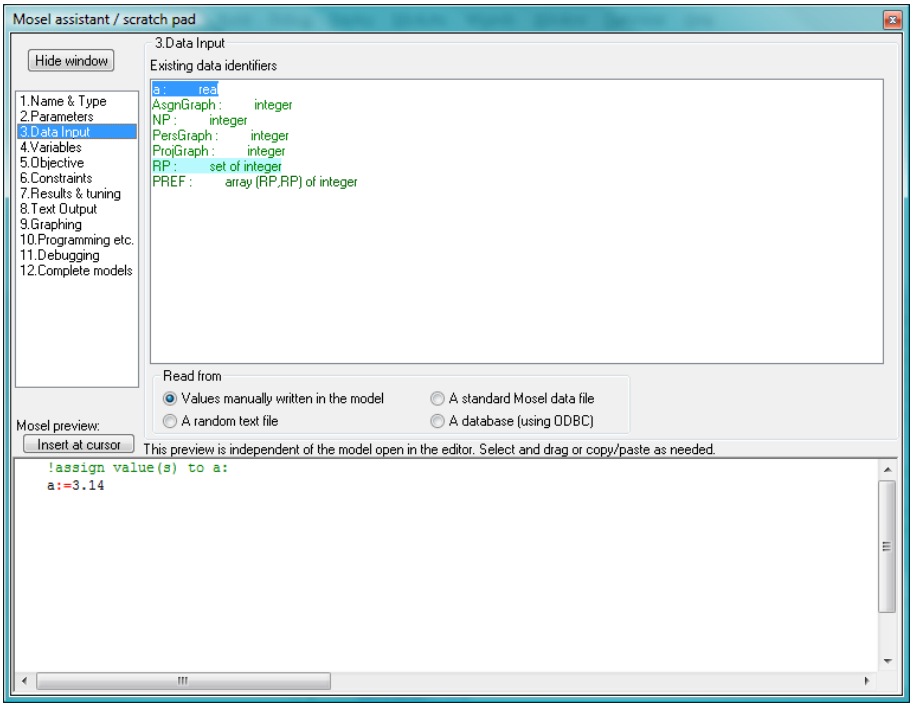
The *Name & Type* wizard assists with creating the skeleton of a model. The skeleton source code is updated to reflect options selected in the wizard window.

5.2 2. Parameters Wizard



The *Parameters* wizard shows a small typical example of using parameters in Mosel. The wizard also shows how to interact with Mosel parameters from a programming language or the Mosel console application.

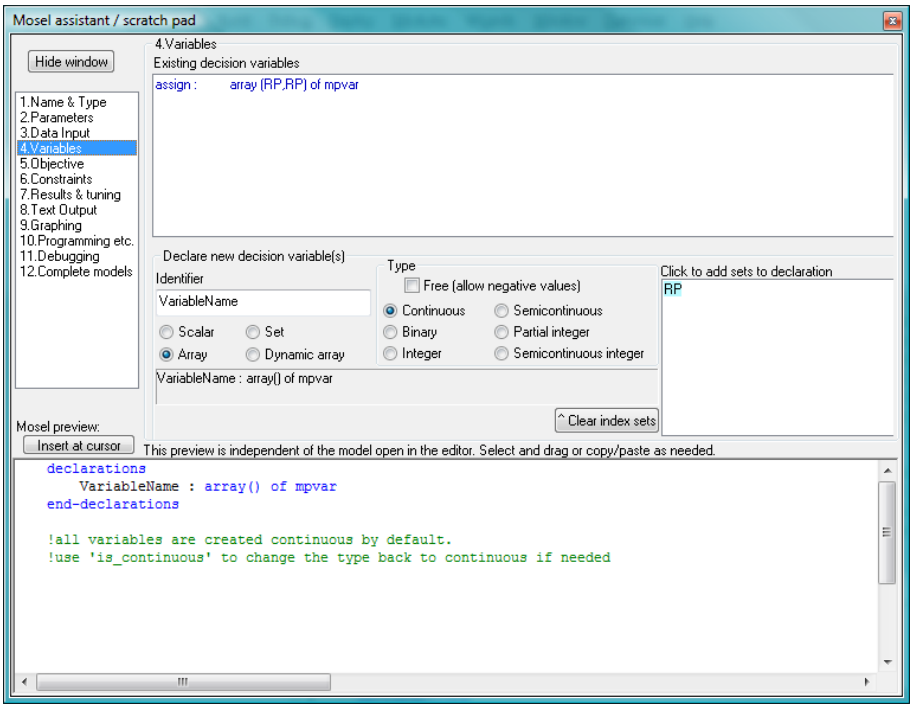
5.3 3. Data Input Wizard



Mosel models can be separated into logic and data. The *Data Input* wizard attempts to obtain all the "data" declarations in a model and then gives an example of how such data may be "read in". Standard Mosel data files as well as reading from random files is supported/described.

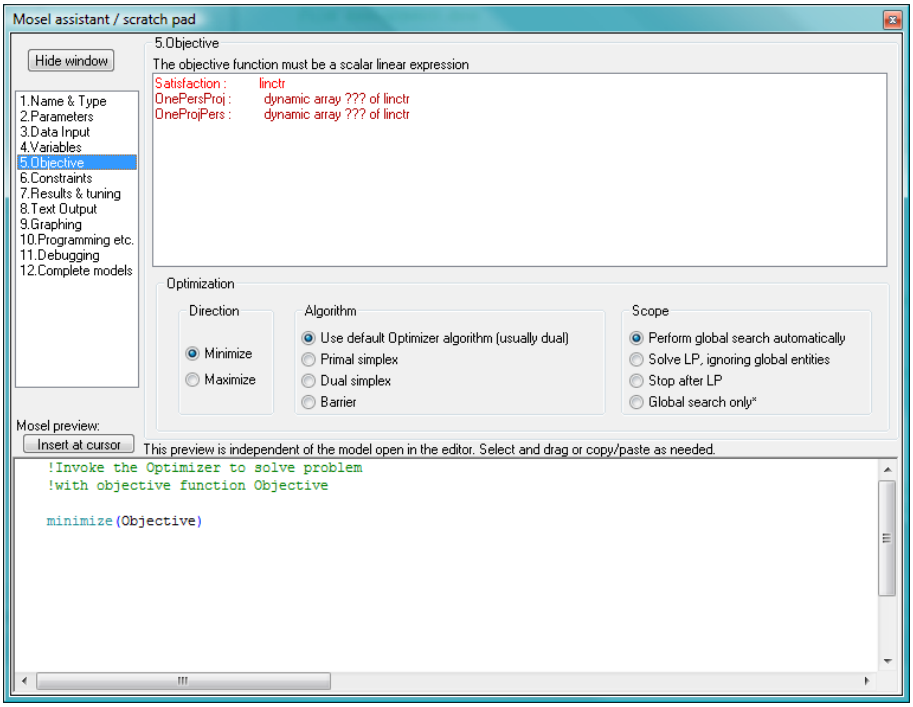
Note that "data" can be of the types integer, boolean, real, string (not mpvar or linctr).

5.4 4. Variables Wizard



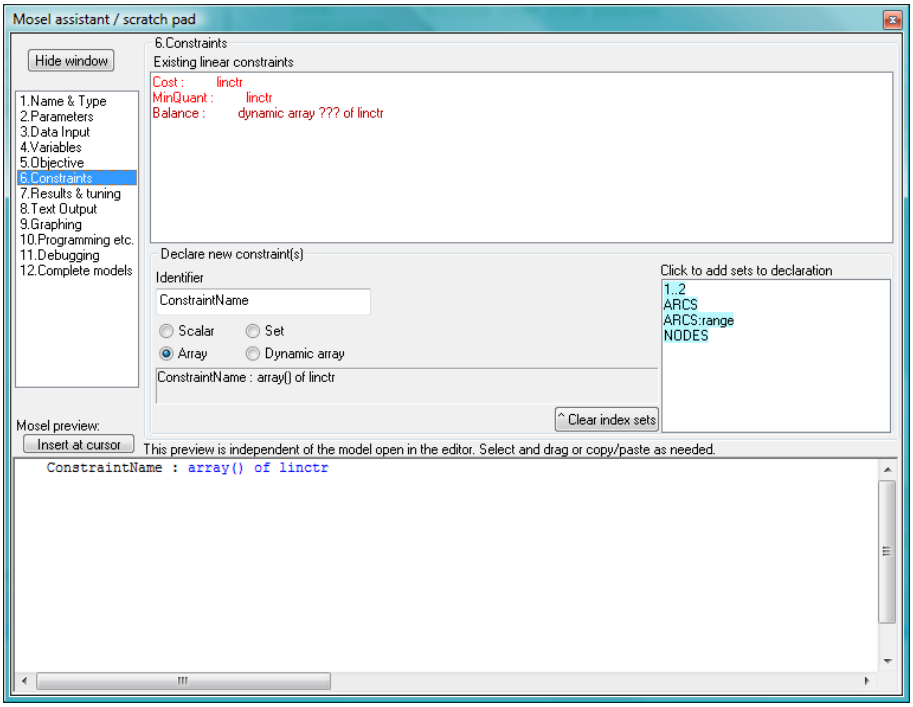
The *Variables* wizard can be used to declare and set the type of decision variables. Type the name of a variable and select index sets (if an array). Optionally, specify the type of the decision variable(s). Watch as the source code is constantly updated based on your selections.

5.5 5. Objective Wizard



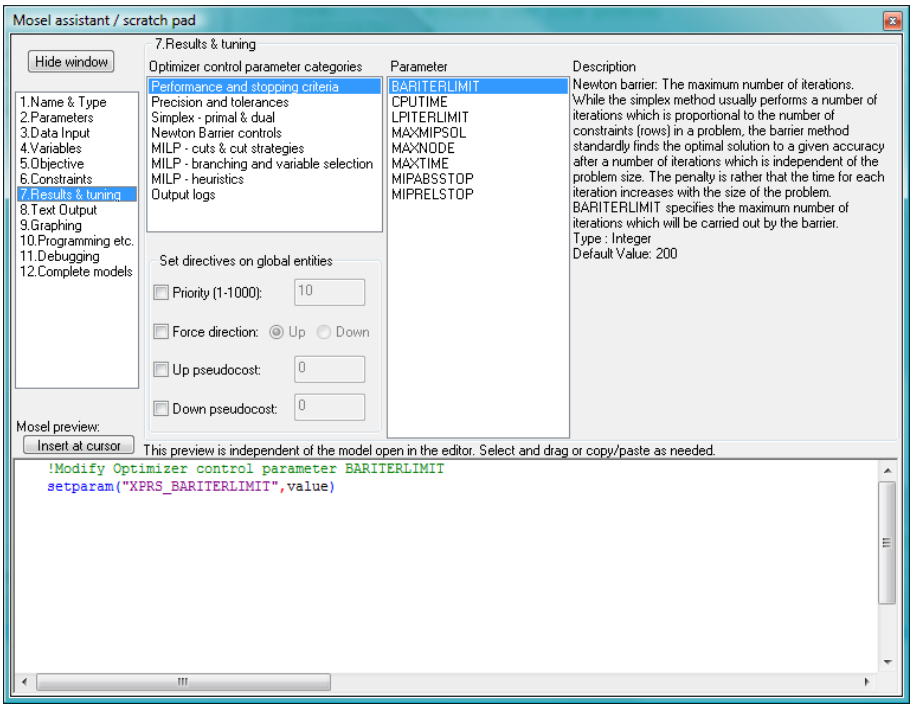
The *Objective* wizard allows the selection of a linctr object as the objective function and explores the various options available when optimizing a model.

5.6 6. Constraints Wizard



The *Constraints* wizard can be used to declare linear constraints. Type the name of a constraint and select index sets (if an array). Optionally, specify the type of the decision variable(s). Watch as the source code is constantly updated based on your selections.

5.7 7. Results & tuning Wizard



The *Results & tuning* wizard has two parts:

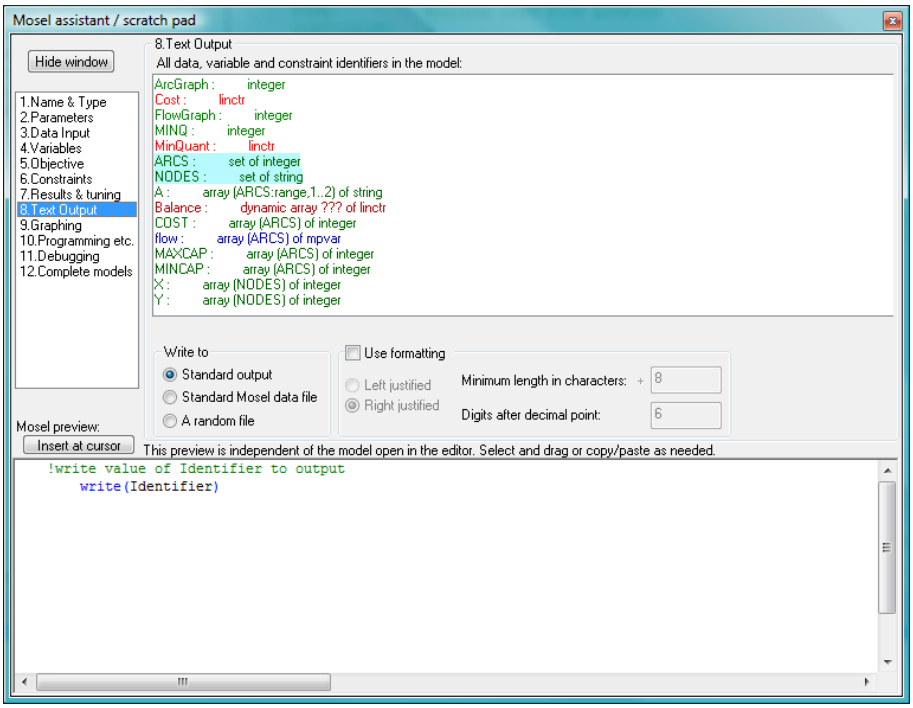
5.7.1 Optimizer control parameters

This interactive section groups all the Optimizer control parameters into families. Select a family and then a control for a full description.

5.7.2 Setting directives on global entities

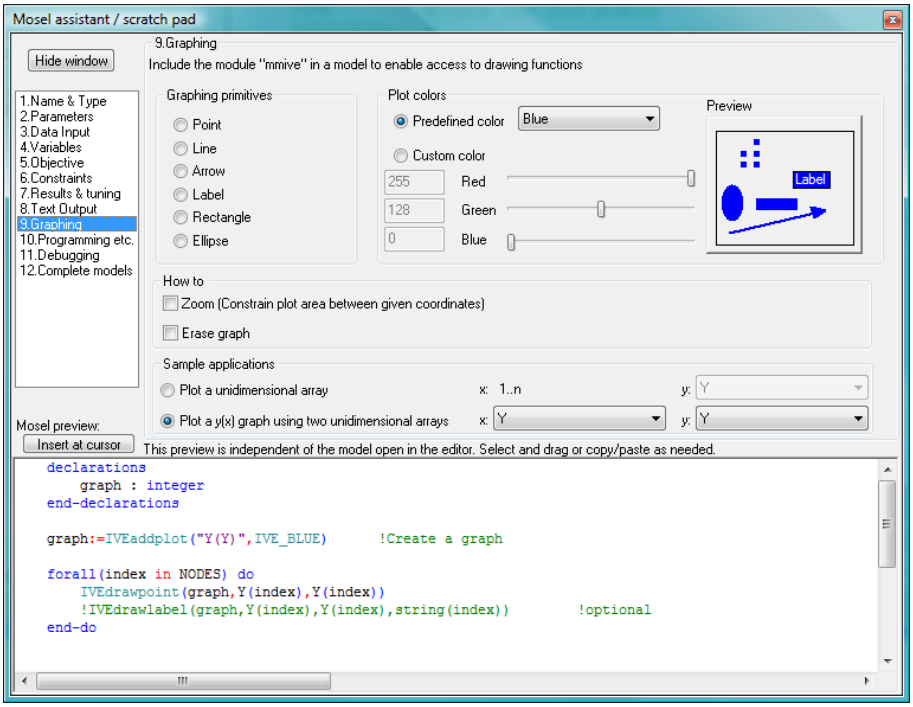
Global entities (integers, binaries, etc.) can be given higher branching priority. Other attributes related to branching can be set for variables.

5.8 8. Text Output Wizard



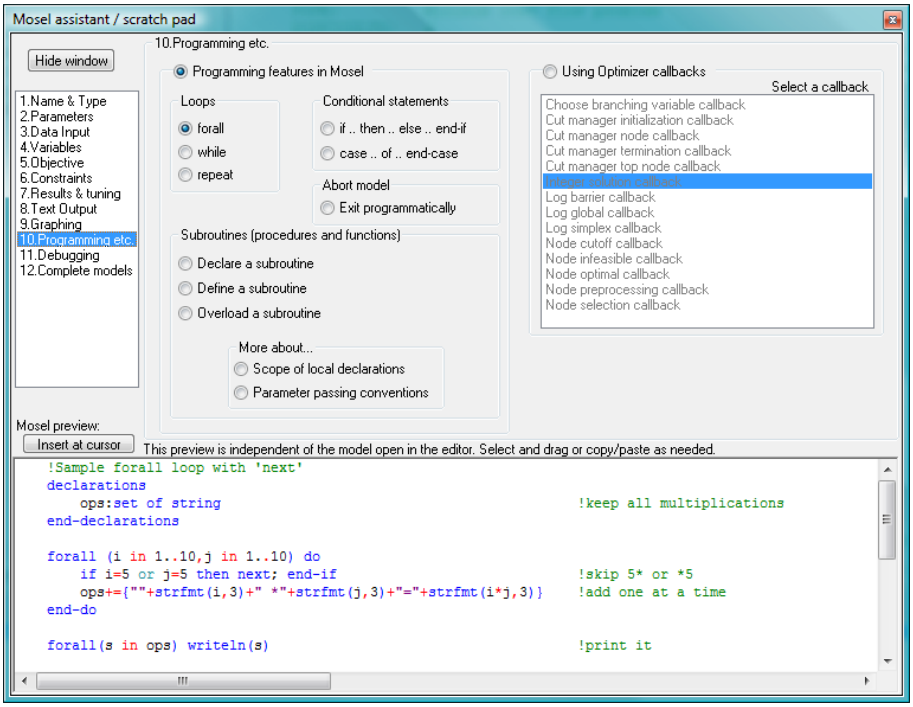
The *Text Output* wizard lists all entities in the current Mosel model and produces code for displaying the values of the entities. Special formatting is optional. This wizard also shows how to write to a file.

5.9 9. Graphing Wizard



The mmive Mosel library can be used for drawing graphs in the User graph section of Xpress-IVE. This wizard explains all the functions/options available for graphing.

5.10 10. Programming Wizard



The *Programming* wizard has two parts:

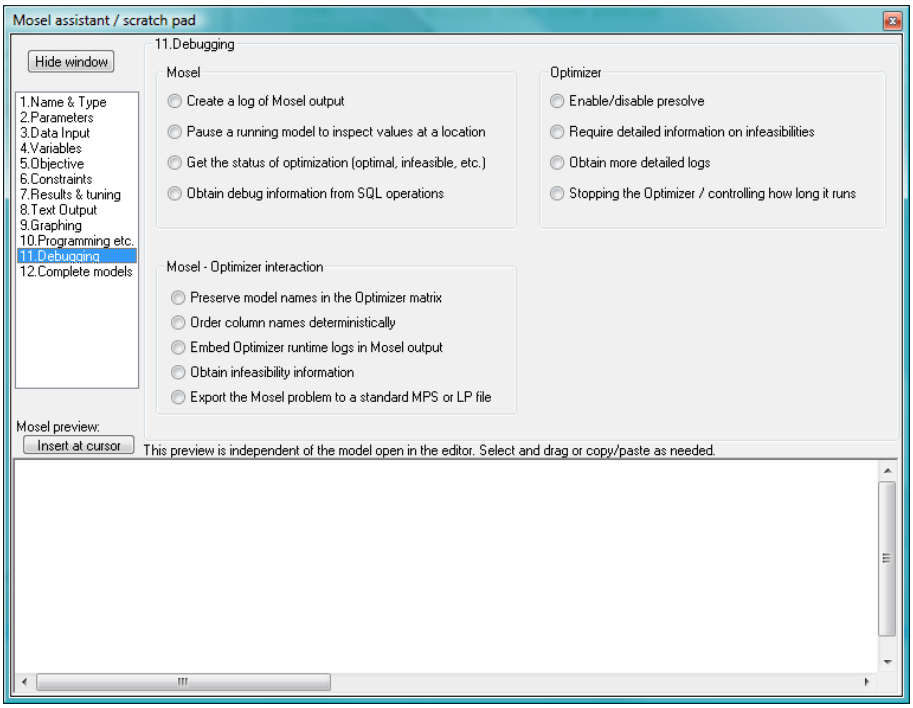
5.10.1 Common Mosel programming tasks

Select a radio button for sample source code that exemplifies that feature.

5.10.2 Setting Optimizer callbacks for advanced interaction with the Optimizer

Check the Xpress documentation for more information on using Optimizer callbacks.

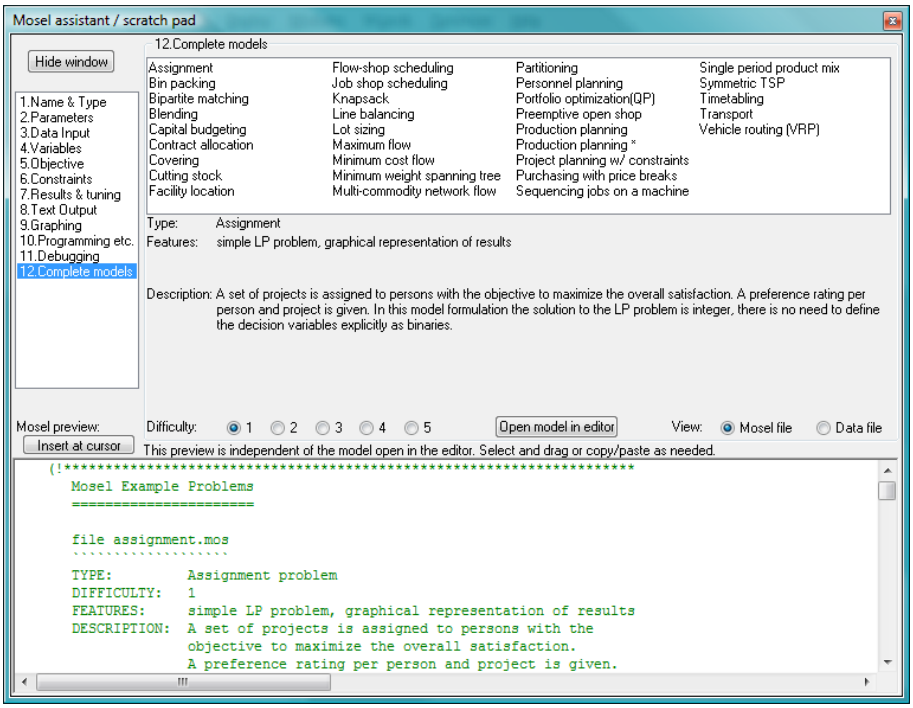
5.11 11. Debugging Wizard



The *Debugging* wizard lists a set of useful debugging features and how to use them. The features are all accesible from Mosel and are grouped by the target of the debugging task:

- Mosel
- Mosel-Optimizer link
- Optimizer

5.12 12. Complete models Wizard



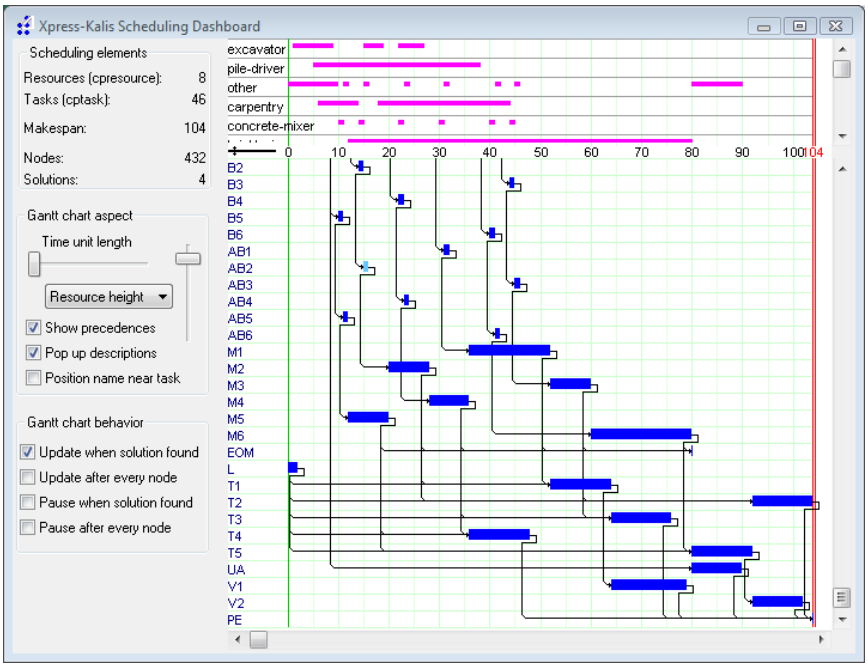
32 complete Mosel models are made available in this version. The models range from simple ones (difficulty:1) to fairly complex (difficulty:5). Select a model type by clicking in the list to obtain more information on the problem it adresses and the Mosel features it employs to solve the problem.

CHAPTER 6

Xpress-IVE Dashboards

Dashboards are special dialogs in Xpress-IVE that monitor the progress and performance of various extensions of the Xpress suite.

6.1 Xpress-Kalis Scheduling Dashboard



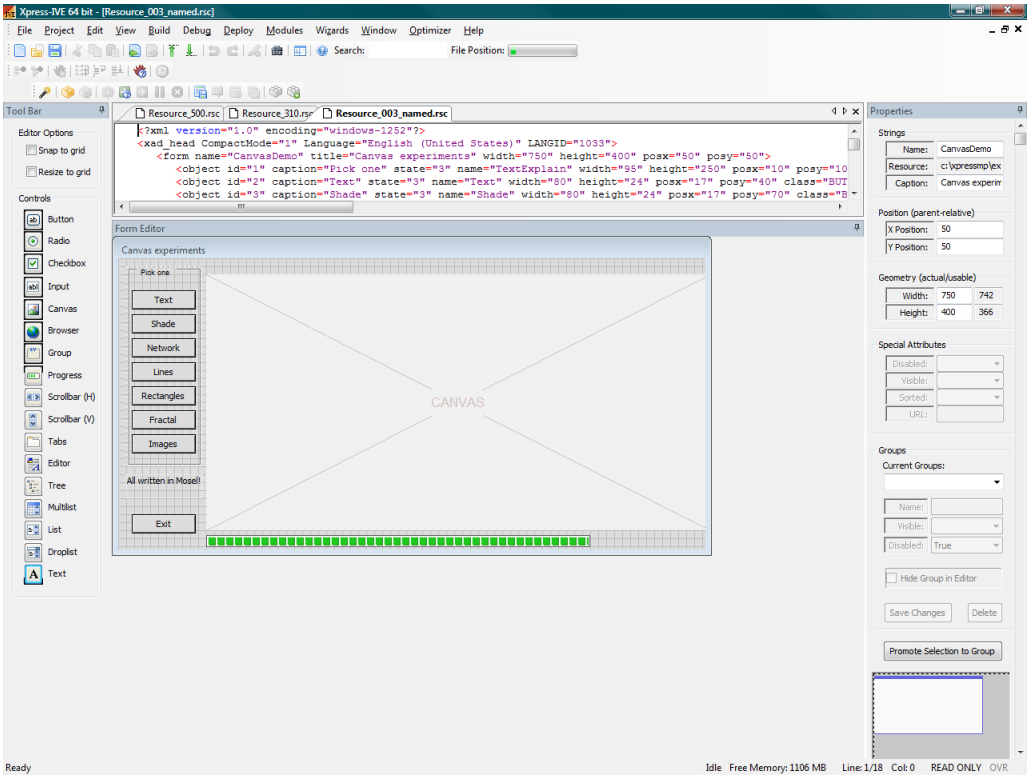
he Xpress-Kalis Scheduling Dashboard is displayed after model execution for models that use the Xpress-Kalis solver (Mosel module "kalis") and contain objects of types `cptask` and `cpresource`. The upper part of the dashboard displays a resource usage chart. The lower half contains a Gantt chart of the scheduled tasks. Arrows between tasks indicate precedence constraints.

CHAPTER 7

XAD resource editor

7.1 Create XAD Forms using a Drag and Drop Interface

The *XAD Resource Editor* can be used to quickly create a form layout for use in XAD Mosel applications. The layout, as defined by the user in the editor, is saved to file in a simple XML format and may then be read in using XAD's XADloadresource function. A full example for doing this from the XAD Editor all the way to a XAD Mosel model can be found on the example page.

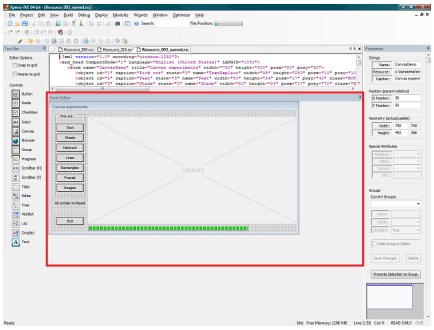


This section contains the following topics:

- The Drag and Drop Toolbar
- The Form Edit Dialog
- The Properties Dialog
- XAD Groups
- The Event Dialog

- Tutorial Example

7.1.1 The XAD Resource Editor Form Edit Dialog



The *Form Edit Dialog* (FED) is used to construct a XAD window object containing XAD control objects through a drag and drop style interface. A form created in this manner may be loaded into a XAD Mosel model and the controls and objects on the form created from the resource file, rather than creating the objects programmatically with Mosel code.

The form itself may be resized through the usual Windows methods of selecting an edge or vertex and left-click dragging, or through an entry in the geometry input controls of the XAD Properties Dialog.

In a similarly Windows fashion the form may be moved around the central XAD editor pane by click-dragging on the form's title bar. Should you wish to move the top of the form beyond the top of the editor pane then you may do so via the form positioning control at the base of the XAD Properties Dialog. The latter method may also be useful if you are editing on a small screen, or with a particularly large form.
Note: To reposition the starting position of the form on the screen when first displayed via XAD Mosel you will need to alter the Position input control of the XAD Properties Dialog. Moving the form within the editor is simply for editing and has no runtime effect.

Once an object is placed on the FED and the properties altered you may wish to add events for the object to a Mosel model file, or create groupings that may be manipulated as one object within your Mosel code.

- **Adding Events:** Event callbacks may be added to a current, or new, Mosel file by double left-clicking on the object you wish to add the event for. This will open up the Events Dialog where you may add, or navigate to, event callbacks within the Mosel code of the specified model. See the Event Dialog page for further details.

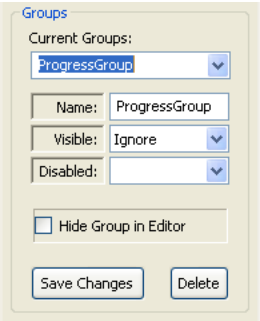
  Note: In order to add an event callback to a file the file must be a valid Mosel model file. Specifically, it must contain the "end-model" line.

- **Select Groups:** To select a group of objects you may hold down Control whilst clicking on the individual controls. You can drag select a box around the desired objects (beginning the selection box anywhere on the form background). Or, if you have already created groups of objects you can select them via the Current Groups control in the XAD Properties Dialog.

- **Delete Objects:** In order to delete the currently selected object you must simply press the Delete or Backspace keys.

- **Repositioning the Form:** You may reposition the form within the editor pane by clicking and dragging as you would a standard Windows window; alternatively, you may use the FED Positioning Control in the XAD Properties Dialog.

7.1.2 XAD Resource Editor Groups



Grouping together objects within the XAD Resource Editor allows multiple objects to be moved, hidden or disabled simultaneously, both in the resource editor itself and, perhaps more powerfully, from within XAD Mosel code.

There are two distinct types of object groups:

Temporary Groups: used simply to move around several objects at a time and keep them at the same relative distances from each other. Temporary groups may be promoted to permanent groups by pressing the Promote Selection to Group button on the XAD Properties Dialog.

Permanent Resource Groups (PRG): permanent group associations of objects. These can be manipulated within XAD Mosel code, hidden or disabled as a group, and allow individual objects to belong to multiple groups (see the 500selectiongroups.mos example discussed in the tutorial). This type of group is stored in the resource file itself and may be loaded in to XAD Mosel code to be associated with a particular group id (see: example).

Groups of objects may be moved by selecting any member object and click-dragging it in the same manner in which you would reposition an individual object on the Form Editor Dialog (FED).

Any selected group of objects will have a bounding box (visible in the image at the top of the page) to display the extremities of the group. Within the FED this denotes the limits of movement of the group. Any attempt to click and drag the group's bounding box beyond the edges of the FED will result in no further movement beyond the FED edge.

Any PRG may be selected, updated or deleted using the Groups control section of the XAD Properties Dialog.

7.1.2.1 Groups Within XAD Mosel Code

Within XAD Mosel you may manipulate and use groups using the following routines:

`XADgroupgetid` Used to retrieve the integer id of a group loaded via a resource file. It takes the arguments `(groupName:string, XADWindowID:integer)`.

`XADgroupgetw` Used to retrieve the integer pixel width of a group of objects. It takes the argument `(groupID:integer)`.

`XADgroupgeth` Used to retrieve the integer pixel height of a group of objects. It takes the argument `(groupID:integer)`.

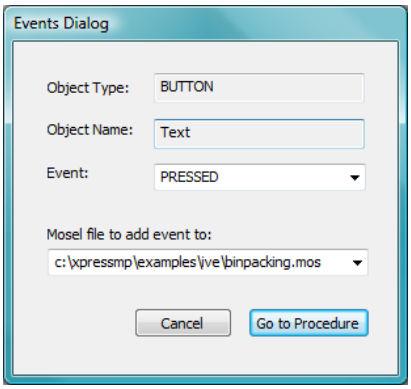
`XADgroupgetx`	Used to retrieve the integer x-position of a group of objects. It takes the argument `(groupID:integer)`.
`XADgroupgety`	Used to retrieve the integer y-position of a group of objects. It takes the argument `(groupID:integer)`.
`XADgroupsetpos`	Used to move the group in unison. The new position sent to the group defines the top left of the group bounding rectangle (remembering that screen-based coordinates have the top left as the origin). It takes the arguments `(groupID:integer, xpos:integer, ypos:integer)`.
`XADgroupenable`	Used to enable or disable all members of a group. It takes the arguments `(groupID:integer, enable:boolean)`.
`XADgroupsetvisible`	Used to show or hide all members of a group. It takes the arguments `(groupID:integer, show:boolean)`.

Additionally, you may also create or destroy groups purely in Mosel code using the following procedures:

`XADgroupcreate`	By creating a set of integers (the XAD ids of those objects you wish to group) you can group them together using this routine. It returns the newly created group id. It takes the argument `(objectIDs:set of integer)`.
`XADgroupaddmember`	Add an object to an already existent group. It takes the arguments `(groupID:integer, objectID:integer)`.
`XADgroupremovemember`	Remove an object from a group. It takes the arguments `(groupID:integer, objectID:integer)`.
`XADgroupdisband`	Remove all objects from a group. It takes the argument `(groupID:integer)`.

Further details of these and other XAD routines may be found in the XAD Reference guide, xadref.pdf.

7.1.3 XAD Resource Editor Event Dialog



This dialog allows the user to quickly add in events for specific objects on the Form Edit Dialog (FED). It is opened by double left-clicking (or right-clicking, if you so desire) on the current object.

Options in the dialog:

Object Type and Object Name: These are read-only and simply indicate the type of the object just selected and the name given to that object (see the XAD Properties Dialog to alter this).

Event: The type of event to add. The selection list will change depending on the type of object currently selected. For instance a BUTTON object will not respond to WINDOW_OPEN events and so it will not appear in the list. A full list of events and object types can be found in the XAD Reference Manual, xadref.pdf, within the Xpress installation "docs" directory.

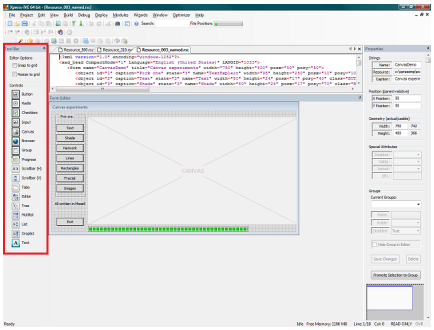
Mosel File to Add Event to: Here you can select any open Mosel code file, choose to open another Mosel file, or choose to create a new Mosel file. The tutorial example creates a new Mosel file, but most commonly you will wish to add events to a current file.

Cancel: Close the dialog without adding an event.

Go To Procedure: This will open the selected Mosel file and, provided it is a valid file, add in the relevant callback and a small piece of code. This code will simply display a message to the effect of "Not yet implemented" and is simply meant as a placeholder until you add your own functioning code to the callback. Note: If the callback already exists in the file then you will be taken to that section of the code and no new code will be created.

Note: If you alter the name of the objects, window or the callback name itself once it has been created then the XAD code will not call that callback during code execution. The callback mechanism works on the assumption that the auto-generated callback will have the name structure "ObjectName_WindowName_EVENT". Should you wish for any reason to alter the name of an object during code execution then you may do so using the XAD procedure `XADsetname(objectID:integer, newName:string)`, but great care should be taken when doing so.

7.1.4 The XAD Resource Editor Drag and Drop Toolbar



This toolbar contains the various controls which may be dragged and dropped on to the Form Edit Dialog (FED) as well as any general options specific to the editor.

The following controls may be selected for easier positioning and resizing of objects within the FED:

Snap to Grid: When repositioning objects with the mouse the object's (0,0) point, or top-left, will position on the FED grid lines. The grid lines are spaced 10 pixels apart in both the vertical and horizontal axes.

Resize to Grid: When resizing objects with the mouse the object's edges, in the direction of resize, will adhere to the FED grid lines. The grid lines are spaced 10 pixels apart in both the vertical and horizontal axes.

To add a control, simply left click on the desired object icon in the left toolbar and, keeping the left button depressed, move the object over the FED. Once the mouse cursor moves over a valid "drop" point the cursor will change from a circular "forbidden" symbol and you may release the mouse button. This creates the representation of the object at the "drop" point and you may then move or resize the control by left clicking and dragging within the centre or the edges of the control, respectively. This behaviour is much the same as standard Windows window behaviour.

The following controls may be dragged and dropped from the toolbar:


Button: The standard Windows button control.


Radio: The standard Windows radio button control. Note that grouping of multiple radio button objects does not cause them to act in a mutually exclusive manner. This must be done programmatically from within XAD Mosel.


Checkbox: The standard Windows checkbox control.


Input: The standard Windows text input control.


Canvas: A canvas object used for drawing on from within XAD Mosel.


Browser: An internet browser window. Set the initial page for the browser using the URL field in the properties dialog.


Group Marker: The standard Windows group identifier box. Note that this has nothing to do with XAD Editor groups.


Progress: The standard Windows progress display.


Scrollbar (Horizontal): The standard Windows horizontal scrollbar control.


Scrollbar (Vertical): The standard Windows vertical scrollbar control.


Tabs: The standard Windows tabbing control. Multiple tabs may be added by separating the tab names with commas in the Caption field of the XAD properties dialog. To create groups of controls related to different tabs within the tab object, create multiple groups and then assign a SELECTION event to the object. Within the SELECTION event in the XAD Mosel code you are then able to hide and disable, or show and enable, the various groups relevant to whichever tab has been selected. Note: A maximum of three tabs within a tab control will display in the editor. This limit does not apply when running the actual XAD Mosel model.


Editor: The standard Windows multiple-line text entry control.


Tree: The standard Windows tree display control.

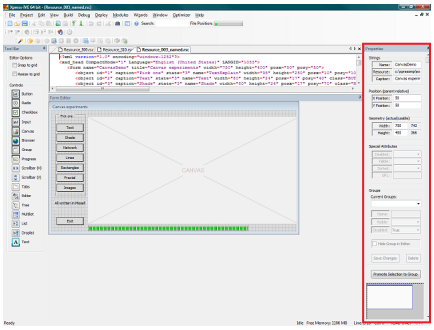
 **Multilist:** The standard Windows multiple-column list control. The list may also have its sorted flag set by setting the Sorted field of the XAD properties dialog to true.

 **List:** The standard Windows list control. Multiple items may be added to the list by entering a comma-separated list within the Caption field of the XAD properties dialog. The list may also have its sorted flag set by setting the Sorted field of the XAD properties dialog to true.

 **Droplist:** The standard Windows droplist control. Multiple items may be added to the list by entering a comma-separated list within the Caption field of the XAD properties dialog. The list may also have its sorted flag set by setting the Sorted field of the XAD properties dialog to true.

 **Text:** The standard Windows read-only text display.

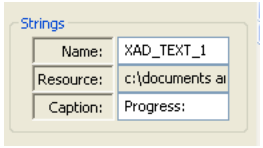
7.1.5 The XAD Resource Editor Properties Dialog



The *Properties Dialog* of the XAD Resource Editor is used to set the various window, object or grouping properies of items on the Form Edit Dialog (FED). In some cases, such as altering size or position, the attributes may be set using click-dragging within the FED.

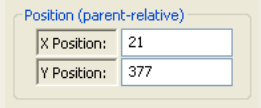
The dialog itself is split up in to several sections, the behaviour of groups deserves a separate page and so is covered in more detail here:

- **Strings**



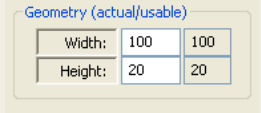
  - *Name:* The internal name given to the object currently selected (used if retrieving the object id from XAD Mosel). This can contain any alpha-numeric character and the underscore character. It should be unique.
  - *Resource:* A read-only field containing the name and full path of the resource currently being edited.
  - *Caption:* If the current object has a text component (the window title in the case of the form) then you may enter it here.

Position (parent-relative)



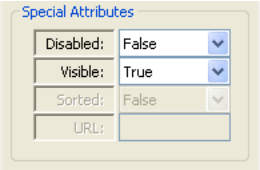
- *X Position:* The x position of the current object within the window; or, in the case of the window itself, the starting x position on the screen when the window is first drawn.
- *Y Position:* The y position of the current object within the window; or, in the case of the window itself, the starting y position on the screen when the window is first drawn.

Geometry (actual/usable)



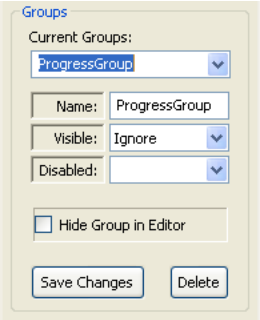
- *Width:* The pixel width of the current object and the relevant size of the usable area. The read-only "usable" text is currently only relevant for the window objects as they have a non-client border area.
- *Height:* The pixel height of the current object and the relevant size of the usable area. The read-only "usable" text is currently only relevant for the window objects as they have a non-client border area.

Special Attributes



- *Disabled:* If you wish the object to be initially disabled (non-responsive to events and in many cases greyed out) then set this control to true. The default is enabled, the control set to false.
- *Visible:* If you wish the object to be initially hidden then set this to false. True by default. When set to false the object will be drawn with a small red X within the editor. To actually hide an object within the editor it must belong to a group and the group itself needs to be hidden using the *Hide Group in Editor* switch.
- *Sorted:* If you wish list objects to be sorted when first displayed then set this to true. False by default.
- *URL:* This string field is currently only relevant for the browser object and should contain the starting address (or home page, to use the common nomenclature) for the browser object when it is first displayed. It will not display anything within the FED object as this is not an actual browser window.

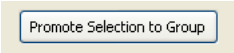
Groups



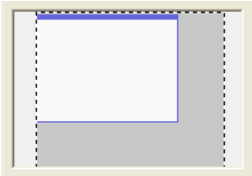
- *Current Groups List:* The list of groups currently associated with the resource. Selecting from the list makes that group the current selection.
- *Name:* The name given to the group, either a default name or a user-given name.
- *Visible:* If you wish the objects within the group to be initially hidden (for instance if you are creating a tab object with multiple tabs) then set this to false. Ignore by default, meaning that the individual object settings for visibility will be honoured. When set to false the objects of the group will be drawn with a small red X within the editor. To actually hide a group of objects within the editor the group needs to be hidden using the *Hide Group in Editor* switch.
- *Disabled:* If you wish the objects of the group to be initially disabled (non-responsive to events and in many cases greyed out) then set this control to true. Ignore by default, meaning that the individual object settings for command response will be honoured. Note that this control will override any previous setting for the individual objects.
- *Hide Group in Editor:* Hide the currently selected group in the editor window. This is only for display in the editor as you are currently working on it and does not affect run-time display of the objects.
- *Save Changes:* Save the changes to the current group (such as a change of name or visibility status).
- *Delete:* Remove the current group from the groups list.

■ **Other controls**

- *Promote Selection To Group:* Clicking this button whilst having a number of objects selected (through shift-clicking, or via drag-selection) will promote that current temporary selection to a saved group which you may then edit via the Groups controls.



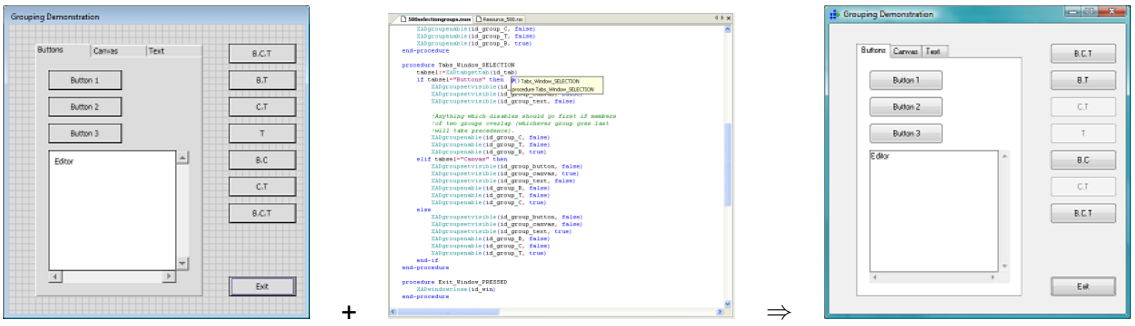
- The *FED positioning control:* This displays the current position and size of the FED within the editing pane area. Should you wish to reposition a particularly large FED you may do so by click dragging the representation of the FED around the control surface. It will not allow you to move the FED to a position completely outside of the positioning control's field of view.



7.1.6 XAD Resource Editor Example

In order to demonstrate the use of the XAD Resource Editor and the associated XAD Mosel commands we will now look at the example "500selectiongroups.mos", in the XAD examples

folder of the Xpress installation. This example not only covers the use of resources, but the manipulation of resource generated groups within Mosel code and the use of an object in multiple groups.



To begin with load the "500selectiongroups.mos" file in to IVE and view the behaviour of the model when run.

In the example the tab object works by picking up the tab's SELECTION event, calling the relevant Mosel callback (Tabs_Window_SELECTION) and then setting the enabled and visible flags of the objects relevant to the currently selected tab. In this case the code required to do this is reasonably simple as we have setup groups of objects which we may hide or show with one command. Herein lies the power of object groups.

We will now look at the various sections of the example's Mosel code, before looking at the associated resource in the XAD Resource Editor. Finally, as a tutorial, we'll create a new resource containing a simple tab object and a few tab-associated controls.

7.1.7 The Mosel Code

500selectiongroups.mos | Resource_500.rsc

```
    XADgroupenable(id_group_C, false)
    XADgroupenable(id_group_T, false)
    XADgroupenable(id_group_B, true)
end-procedure

procedure Tabs_Window_SELECTION
    tabsel:=XADtabgettab(id_tab)
    if tabsel="Buttons" then
        XADgroupsetvisible(id_
        XADgroupsetvisible(id_
        XADgroupsetvisible(id_group_text, false)

        !Anything which disables should go first if members
        !of two groups overlap (whichever group goes last
        !will take precedence).
        XADgroupenable(id_group_C, false)
        XADgroupenable(id_group_T, false)
        XADgroupenable(id_group_B, true)
    elif tabsel="Canvas" then
        XADgroupsetvisible(id_group_button, false)
        XADgroupsetvisible(id_group_canvas, true)
        XADgroupsetvisible(id_group_text, false)
        XADgroupenable(id_group_B, false)
        XADgroupenable(id_group_T, false)
        XADgroupenable(id_group_C, true)
    else
        XADgroupsetvisible(id_group_button, false)
        XADgroupsetvisible(id_group_canvas, false)
        XADgroupsetvisible(id_group_text, true)
        XADgroupenable(id_group_B, false)
        XADgroupenable(id_group_C, false)
        XADgroupenable(id_group_T, true)
    end-if
end-procedure

procedure Exit_Window_PRESSED
    XADwindowclose(id_win)
end-procedure
```

P() Tabs_Window_SELECTION
procedure Tabs_Window_SELECTION

The Mosel code has the following parts (ignoring those parts common to standard Mosel models):

- Load the window from resource:
  Here we load the resource file in to the model. All resources equate to one XAD window and the return value of the function used to load the resource, XADloadresource, is the id of the XAD window object (id_win, in this case).
- Retrieve the object/group ids:
  Although we need not retrieve the object ids for all of the resource objects, if we wish to manipulate or respond to events for that object we must do so. When creating the resource each object/group will have been given a name (either the default "XAD_OBJECTTYPE_COUNTER", or set by the user) and it is this that we will use to retrieve the object ids.

Using the XAD functions `XADgetid` and `XADgroupgetid` we may retrieve the ids for objects and groups, respectively.

- Display the window:
  This function opens the specified XAD window and displays all the associated objects. In order to only display/enable those objects relevant for the initially displayed tab we will need to setup the object states when the window opens. This is achieved via a WINDOW_OPENED event callback.
- procedure Window_WINDOW_OPENED:
  In this callback we need to setup the various states of the objects/groups belonging to each tab selection. There are six groups within the example, 3 relating directly to those objects displayed on each tab, and 3 relating to the buttons on the right hand side of the example.

  The right hand side buttons demonstrate that when the Button, Canvas or Text tabs are selected the relevant buttons are enabled or disabled. These differ from the groups setup for the objects in the tab control as each button may belong to more than one group (Group B, C or T depending on which tab selections they will be enabled for).

  We intially have the "Button" tab selected and so within this callback we enable the id_group_button and id_group_B groups and disable the others.

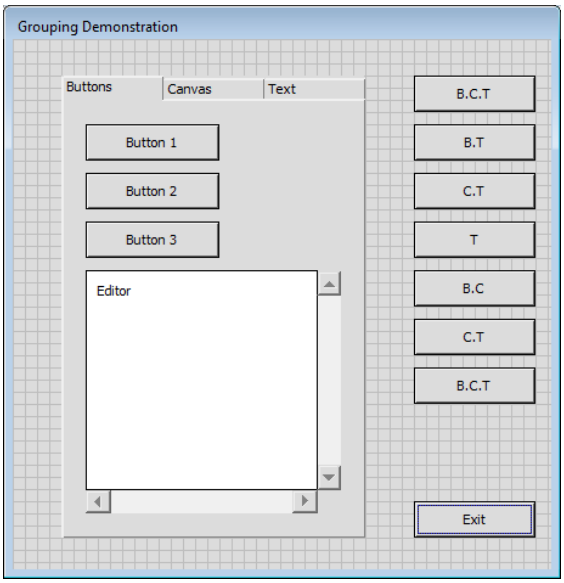
  Note: there are no events associated with the buttons in this example and so they will not actually perform any action if clicked.
- procedure Tabs_Window_SELECTION:
  This callback is in essence very similar to the WINDOW_OPENED callback, above. The difference being that we must check for the currently selected tab and then disable/enable and show/hide the relevant groups for each tab.

  As the comment in the code mentions, it is important to get the order of the commands correct if you are dealing with objects in multiple groups. Were the command order incorrect you may inadvertently enable and then disable an object (belonging to multiple groups) that you intended to be enabled.

  Note: It is recommended to first disable all the groups you need to before finally enabling the relevant group or groups (as in this example).
- procedure Exit_Window_PRESSED:
  When the "Exit" button is PRESSED this callback is called. All it does is cleanly close the id_win window so that the program closes in a user controlled and clean manner.

7.1.8 The Associated Resource File

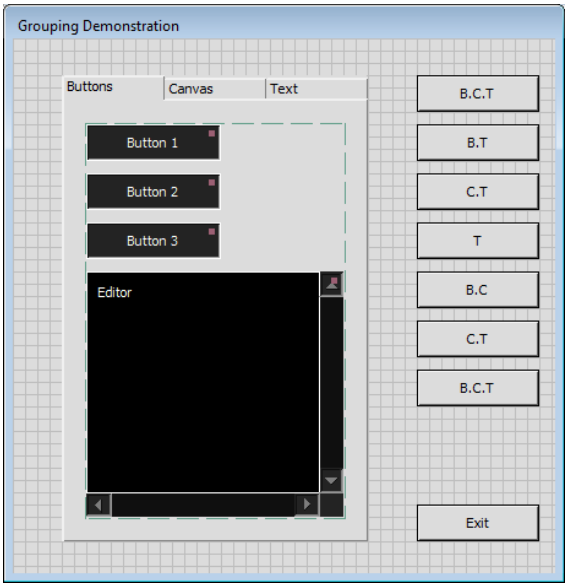
The resource file associated with the example, "Resource_500.rsc", can be found in the XAD examples directory alongside the Mosel file. Once loaded in to IVE you will be presented with the XAD Resource Editor and the representation of the XAD window and objects will be visible in the Form Edit Dialog (FED).

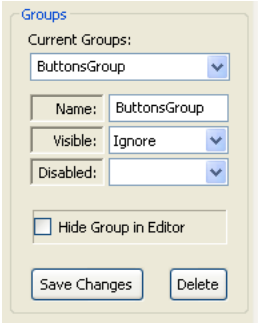


When initially loaded all of the objects within the tab will be visible. To hide a group of objects within he editor select the group from the XAD Properties Dialog Group drop-down list and then select the option to Hide Group in Editor.

This behaviour can be used to quickly shift between group selections designed for use in tab objects. In this example we will hide the Buttons group and show the Canvas group:

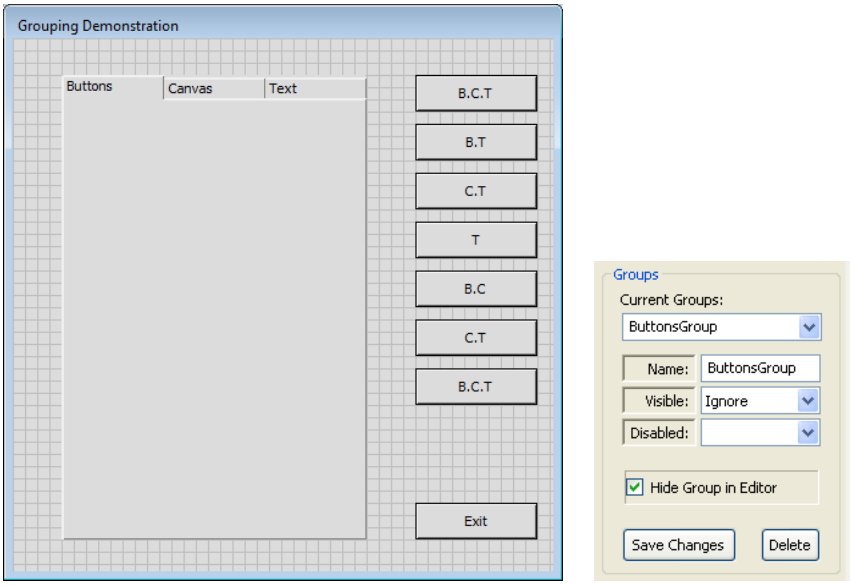
- Select the group you wish to hide:





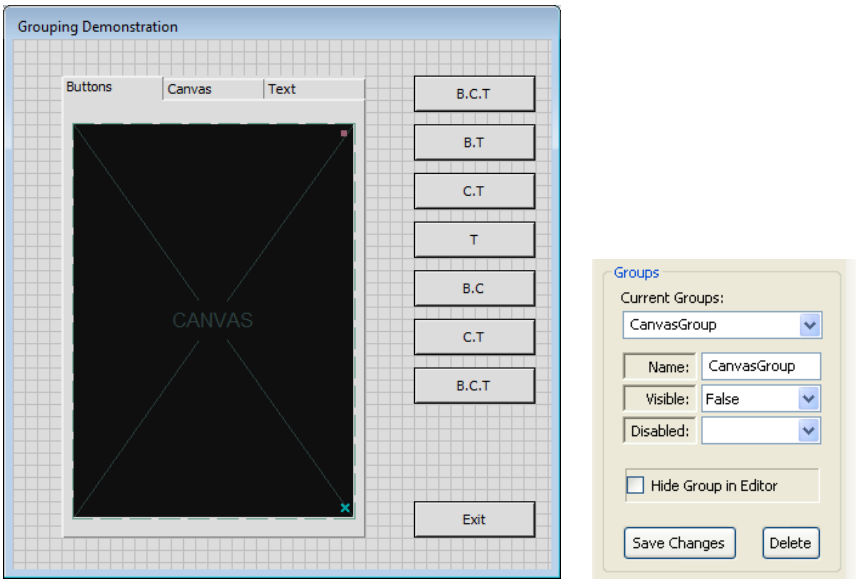
Here we've selected the Buttons group. It's currently visible in the editor.

- Hide the Buttons group:



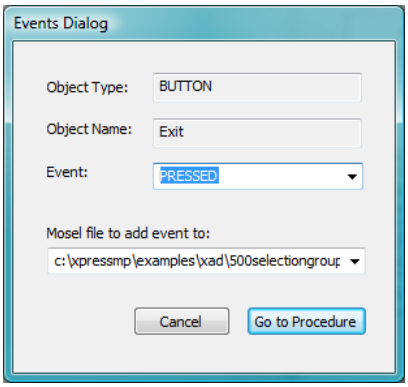
By selecting the Hide Group in Editor option we hide the Buttons group.

- Unhide the Canvas group:



By selecting the Canvas group in the drop-down list, and unselecting the Hide Group in Editor option, the Canvas group becomes visible in the editor.

The events for the objects on the resource may be added, or navigated to, via the *XAD Event Dialog*. This is shown when an object, or the form itself, is double left-clicked.



As an example we will now navigate to the *Exit* button callback discussed earlier. To do so, firstly double click the *Exit* button in the FED to open the *Event Dialog* for the button. Once this is open we can navigate to the event callback in the following manner:

- Select the Event:
  In this case we wish to select the PRESSED event, but were you adding a different event you could select any of the events offered to you in the drop-down list.
- Select the Mosel File:
  We wish to navigate to the event callback already set in "500selectiongroups.mos", but you could choose to add the event to any valid Mosel file in which you intend to load the "Resource_500.rsc".
- Go to Procedure:
  Once you've selected the event and file you wish click the button and you will be taken to the relevant callback in the file specified. In this case the callback already exists and so you should now see the XADwindowclose function call which forms the operational code of the callback.

  If you'd chosen to add a currently non-existent event callback to the file then the code part of the callback would contain the default "Not yet implemented" Mosel text output.

7.1.9 Creating a Simple Tab Example from Scratch

In this section we will use the Resource Editor to create a simple tabbing application for XAD from scratch, using the drag and drop features of the editor to layout the form and the Event Dialog to add in a few simple events.

Firstly, create a new resource by selecting the New Resource option on the File menu. This will create a blank form in the FED of the default size (500x500).

Now we'll alter the form to the desired size and set its various attributes:

- Resize:
  By clicking and dragging on any side or vertex of the form (or by entering the size in the Properties Dialog) resize the form to around 600x400. It's recommended that if you want a very specific size you set this via the Properties Dialog, otherwise using the mouse to resize can be quicker and easier.
- Setup Strings:
  Next you will need to enter a name for the form (XAD window) object. You may stick with the default "XAD_WINDOW_NUMBER" if you wish, but you may find it easier to maintain your Mosel program if you give it a more unique name. For this example you could enter "Main_Window".

Enter a caption for the window (the title which appears in the form's top bar) via the Properties Dialog. For this example you could enter "Tutorial Window".

- Set the Window Position on Screen:
  For this example we'll have the window open and display at half its own width and height from the top left of the screen. If you've set a form width and height of 600x400 then set the "X Position" to 300 and the "Y Position" to 200.

Next we'll add a button object for closing down the program once it's started. This will involve adding the button itself via the drag and drop editor, creating the new Mosel model file and adding the PRESSED event to close the window in the Mosel code:

- Adding the Button Object:
  Click and drag a button object from the drag and drop toolbar on the left-hand side of the editor. Once your mouse cursor moves over a "droppable" area of the newly created form you can release the mouse button; this will create a button at the position of the mouse cursor. To refine the position of the button you may left-click and drag it within the form, or you may enter the position in the Properties Dialog, as we did with the main form. In this example we'll place the button towards the bottom right of the form.
- Modify the Button Attributes:
  As with the main form we should give the newly created button a more descriptive caption and, if desired, a more unique name. In this example we will mark the button as the "Exit" button and give it the name "Exit".
- Add a PRESSED Event:
  To make the button function when pressed we need to add an event via the Event Dialog. With the "Exit" button selected (its attributes will be visible in the XAD Properties Dialog), double left-click on the button to open the Event Dialog. Select the PRESSED event, as we want this callback function to be called when the button is pressed, and make sure "Create a new file" is selected as the Mosel file to add the event to. Now click "Go to Procedure" and enter a location and name for your new Mosel file. For this example we'll call the new Mosel file "Tutorial.mos".

  You should now see the newly created Mosel file with the Exit_Main_Window_PRESSED procedure inserted. As we want the procedure to close the window replace the default code with XADwindowclose(id_win) as in the original example above.

  In this instance, as we've created a new Mosel file, the code to load the resource and assign this particular resource to the id id_win isn't yet present. Had we added this to an already existent Mosel file then we would of course close the window of whatever XAD id had been associated with the resource.

To create a functioning application we now need to add in a little connecting code to load this particular resource file and open the associated window:

- Enable XAD Code Usage:
  As we'll be using the XAD Mosel module we'll need to add the line "uses "mmxad"" to the Mosel code.
- Declare the Window ID:
  Within the declarations section of the Mosel code you will need to declare the id of the XAD window we will associate with the resource. In this case we're calling the id id_win, as above. All ids are simply integer values and therefore we must add the line "id_win: integer".

- Load the Resource: To associate a resource with an id the resource (and thus its associated objects) are loaded using the XAD function "XADloadresource". This function returns the integer id it has associated with the loaded resource and we assign this to id_win.

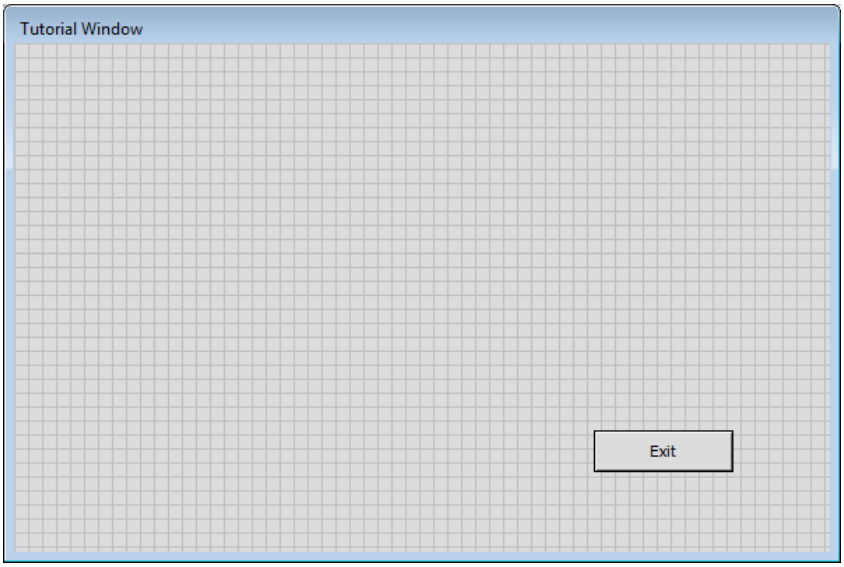
  Note: if your Mosel code file and resource file are saved in different folders then you will need to specify a path to the resource file in the argument to "XADloadresource".

- Open the Window and Begin Execution:
  In order to hand control of the application to the "Main_Window", id_win, we need the program to display the window and associated controls. To do so use the XAD function "XADwindowopen(id_win)".

- Exit the Program:
  To cleanly exit and end the main section of the code we now add the "exit(0)" command to the Mosel code.

- Cleanup the Code:
  Whenever a new Mosel file is created in IVE there may be default sections that are inappropriate to your program. In this case we do not require the reference to "mmxprs", the "parameters" section or the Mosel "writeln" code and so they can be removed.

The Mosel code and resource should now look like similar to the images below and you should be able to run the application in IVE. Try running the program and notice that we can exit using the button and event we've created.




```
model ModelName
uses "mmxad"

declarations
  id_win: integer
end-declarations

id_win := XADloadresource("Tutorial.rsc")

XADwindowopen(id_win)

exit(0)

procedure Exit_Main_Window_PRESSED
   XADwindowclose(id_win)
end-procedure

end-model
```

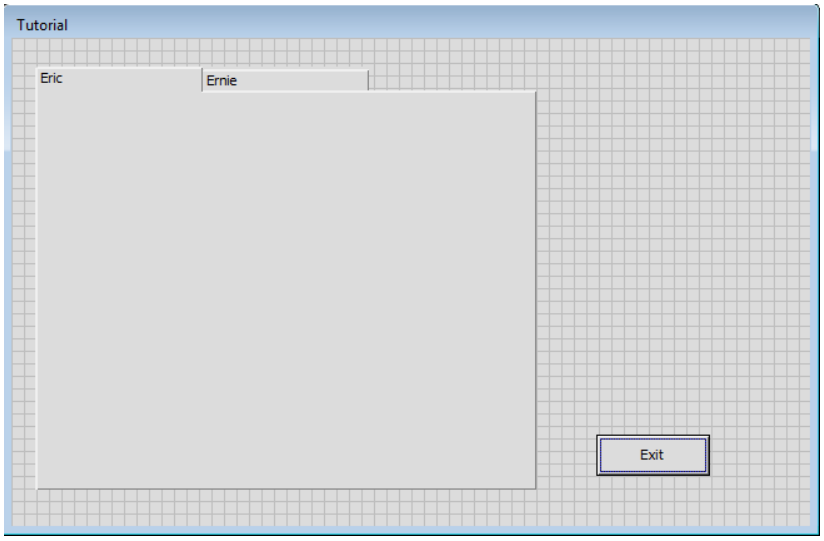
Finally, we'll add the tab object with a couple of XAD objects associated with each tab:

- Add the Tab Object:
  In the same manner in which you added the button, previously, add a tab object to the form. Again, in the same manner as the button, give it the name "The_Tab" and position it towards the top left of the form. Now, in the same manner as you resized the form object at the start of the tutorial, resize the tab object to fill the form to the right of the "Exit" button.

- Create Multiple Tabs:
  Instead of having a single caption or title, the tab object has tab captions to the top of each tab. These are added in the same manner as the caption added to the "Exit" button, but with the tab captions delimited by the comma character.

  In the XAD Properties Dialog enter "Eric,Ernie" in the Caption field. This creates two tabs on the tab object. A maximum of three tabs will be visible on the FED, although you may enter as many comma-separated tab caption entries as you wish in the Caption field.

  The resource editor should now appear similar to the following:



- Add Objects for Each Tab:
  On each tab we want different objects to appear. In this example we will create canvas and text objects on the "Eric" tab and a web browser on the "Ernie" tab:

- In the same manner as the other drag and dropped objects, add a browser object to the form and reposition/resize it to sit over the tab object. You will need to drop the browser on a section of the form itself.

- Next set a name for the browser, "Browser" in this example, and set the URL field to the web page you wish it to open on when clicked, www.fico.com in this example.

- All objects associated with a tab should be grouped together. For the "Ernie" tab the group consists of the single browser object, "Browser".

  Whilst holding down the Control key left click on the browser object. This creates a selection group.

  To make this a permanent group, for use in XAD Mosel code, you now need to promote this to a group using the Promote Selection To Group button of the XAD Properties Dialog. This creates a group called "Group_1".

  Rename this newly created group to something more unique, such as "Group_Ernie" and set the Visibility to False and Disabled to True as initially the "Eric" tab will be visible. Click "Save Changes" to save the group updates. All of the commands to rename and update groups are to be done via the Groups section of the XAD Properties Dialog.

Now hide the browser object in the editor so that we can add the "Eric" tab objects. With "Group_Ernie" selected click the "Hide Group in Editor" checkbox on the XAD Properties Dialog.

- For the "Eric" tab add the canvas and text objects in the usual drag and drop manner. For the sake of keeping the Mosel example code simple in this example the canvas object will simply be blank. The text object may contain any text you wish and may be entered via the Capion field of the XAD Properties Dialog. In this example the objects have been given the names "Eric_Canvas" and "Eric_Text".

  By Control clicking the "Eric_Canvas" and "Eric_Text" objects create another group called "Group_Eric". This group should be visible and enabled.

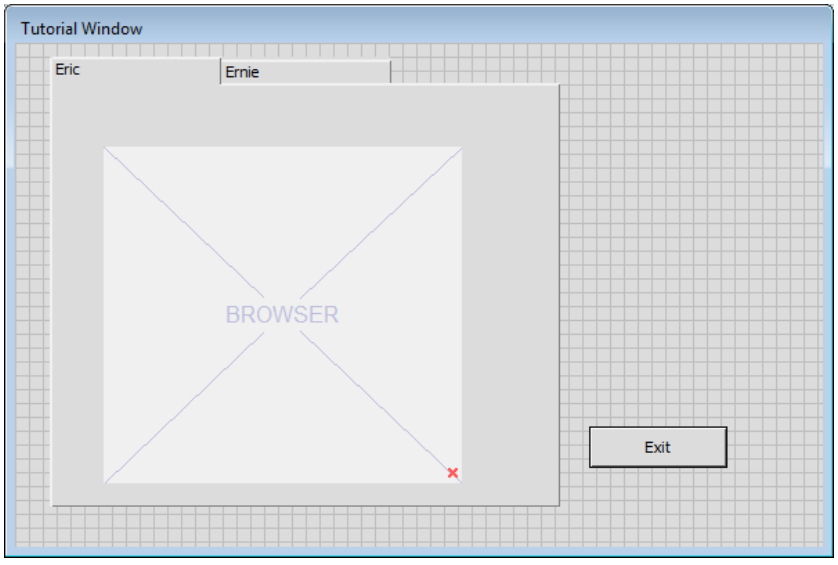
- Add the SELECTION Event Callback:
  Now that the groups of objects are setup we should add the tab SELECTION event callback to the Mosel code so that something actually happens when the tab with focus is changed. Double left-clicking on the "The_Tab" object open the Event Dialog, select the SELECTION event and the Mosel file you created previously and "Go to Procedure".

- Associate Code-Referenced Objects with Resource Objects:
  Any objects, or groups, created in the Resource Editor that you wish to manipulate in the Mosel code must have an associated id. In this example we will reference the "The_Tab" object and the two groups "Group_Eric" and "Group_Ernie". To associate an integer id with these objects use the two XAD Mosel functions XADgetid and XADgroupgetid, both of which take the string name of the object or group and the id of the XAD window it is associated with, id_win in this case.

- Add the Tab Selection Handling Code:
  When either of the two tabs is selected we want to show and enable the objects associated with it and hide and disable those objects not associated with it.

  Within the The_Tab_Main_Window_SELECTION procedure use the XAD Mosel function XADtabgettab(id_tab) to retrieve the string name of the currently selected tab. Compare this name to the two tab names "Eric", or "Ernie" and enable/show or disable/hide the two groups as appropriate using XADgroupsetvisible and XADgroupenable.

  This should result in the following resource and Mosel code:



```
model ModelName
uses "mmxad"

declarations
  id_win: integer
  id_tab: integer
  id_group_eric: integer
  id_group_ernie: integer
end-declarations

id_win := XADloadresource("Tutorial.rsc")

id_tab := XADgetid("The_Tab", id_win)
id_group_eric := XADgroupgetid("Group_Eric", id_win)
id_group_ernie := XADgroupgetid("Group_Ernie", id_win)

XADwindowopen(id_win)

exit(0)

procedure Exit_Main_Window_PRESSED
   XADwindowclose(id_win)
end-procedure

procedure The_Tab_Main_Window_SELECTION
    tabsel:=XADtabgettab(id_tab)
    if tabsel="Eric" then
        XADgroupsetvisible(id_group_ernie, false)
        XADgroupenable(id_group_ernie, false)
        XADgroupsetvisible(id_group_eric, true)
        XADgroupenable(id_group_eric, true)
    else
        XADgroupsetvisible(id_group_eric, false)
        XADgroupenable(id_group_eric, false)
        XADgroupsetvisible(id_group_ernie, true)
        XADgroupenable(id_group_ernie, true)
    end-if
end-procedure

end-model
```

You have now created a very simple working example using the resource editor, groups and events. Provided you have saved all changes to the resource file you should now be able to run the example and switch between the tabs successfully.

Index