

MPL Modeling System

Release 4.2

MPL Modeling System

Release 4.2

Maximal Software, Inc.
2111 Wilson Boulevard
Suite 700
Arlington, VA 22201

Tel: (703) 522-7900
Fax: (703) 522-7902
Email: info@maximalsoftware.com
Web: www.maximalsoftware.com

Copyright © 1988-2002, Maximal Software, Inc., All rights reserved.
CPLEX dialog box items: Copyright © 1989-2002, CPLEX Optimization, Inc.
XPRESS dialog box items: Copyright © 1984-2002, Dash Optimization, Ltd.

No part of this document may be reproduced, stored in a retrieval system, translated, or transmitted by any means, electronic, mechanical, or otherwise, without the prior written permission of Maximal Software, Inc.

Printed in USA 2002 - MUMPLW/021

The information in this document is subject to change without notice and does not represent a commitment by Maximal Software, Inc.

Maximal™, MPL™, and Turbo-Simplex™ are trademarks of Maximal Software, Inc.
Microsoft®, and Windows® are registered trademarks of Microsoft Corporation.
Linux® is a registered trademark of Linus Torvalds.
CPLEX® is a registered trademark of ILOG
XPRESS™ is a trademark of Dash Optimization, Ltd.
IBM® is a registered trademark of International Business Machines Corp.
XA™ is a trademark of Sunset Software Technology.
Frontline™ is a trademark of Frontline Systems, Inc.
Lindo® is a registered trademark of Lindo Systems, Inc.

PREFACE

Welcome to the *MPL Modeling System*, a full-featured modeling system for optimization running under Windows and Motif. Never before has an optimization modeling system software offered such advanced features, yet been so easy to use.

Maximal's philosophy has produced a unique product that has both speed and efficiency. The user interface is particularly friendly and natural to use; our goal was to make the input form resemble, as closely as possible, how you would write out a problem on paper. Making the formulation of optimization models both straightforward and easy. **MPL** is equally at home in a university setting as it is in the business world.

This manual is intended for the owner of the *MPL Modeling System* package. It includes reference information for people experienced in developing optimization models and help for those just learning about it.



Contents at a Glance

PART I	GETTING STARTED	1
Chapter 1	Introduction.....	3
Chapter 2	Getting Started With MPL.....	9

PART II	USING THE MPL MODELING SYSTEM	13
Chapter 3	Running MPL	15
Chapter 4	Description Of Menus.....	35

PART III	THE MPL MODELING LANGUAGE	141
Chapter 5	Language Overview	143
Chapter 6	Defining Sets And Indexes	155
Chapter 7	Data For The Model.....	167
Chapter 8	Formulating The Model	177
Chapter 9	Building Formulas	195
Chapter 10	Advanced Indexing Techniques.....	209
Chapter 11	Database Connection	217

PART IV	A MPL TUTORIAL	229
Tutorial Overview		231
Session 1: Running MPL on a Sample Model.....		233
Session 2: Formulating a Simple Product-Mix Model		243
Session 3: Introducing Vectors and Indexes in MPL Models		255
Session 4: A Production Planning Model with Multiple Time Periods		269
Session 5: A Production Planning Model with Multiple Plants		283
Session 6: Upgrade the Model to Allow Shipments Between the Plants		295
Session 7: Formulating Models With Sparse Data in MPL.....		309

APPENDICES		323
Appendix A: Character Set.....		325
Appendix B: Error Messages		327



Table of Contents

PART I GETTING STARTED 1

CHAPTER 1 INTRODUCTION 3

- 1.1 What Is MPL?.....4
- 1.2 Key Features of MPL.....5
- 1.3 How to Contact Maximal Software7

CHAPTER 2 GETTING STARTED WITH MPL 9

- 2.1 System Requirements10
- 2.2 Installing MPL for Windows10
- 2.3 Setting up Solvers for MPL11
- 2.4 Using the MPL Environment.....12

PART II USING THE MPL MODELING SYSTEM 13

CHAPTER 3 RUNNING MPL 15

- 3.1 How to Start MPL.....16
 - Running MPL from the Start Menu16
 - Create a Shortcut for MPL.....16
 - Embedding MPL Models in Applications17
 - Running MPL in Run-time Mode17
 - Calling MPL from Other Applications18
- 3.2 The MPL Main Window.....19
 - The Main Menu19
 - The Pull-down Menus20
 - The Toolbar22
 - The Status Bar23

3.3 Using the MPL Environment.....	24
Solver Support in MPL.....	25
Solving Models in MPL.....	26
The Model Definitions Window	28
The Message Window	31
Using Projects to Manage Models.....	32
Using the MPL Help System	33
Context Sensitive Help	34

CHAPTER 4 DESCRIPTION OF MENUS 35

4.1 The Main Menu	36
4.2 The File Menu	37
Create a New Model File.....	37
Open an Existing Model File.....	38
Close Files	39
Save the Model File.....	40
Save Selected Text to a File.....	41
Insert File Into the Editor.....	42
Print the Model File.....	43
Exit the MPL Program.....	44
4.3 The Edit Menu.....	45
Undo Changes	45
Clipboard Operations.....	46
4.4 The Search Menu.....	47
Find Text in the Model	48
Find and Replace Text	49
Goto Line in the Model.....	50
4.5 The Project Menu	51
Create a New Project File.....	52
Open an Existing Project File.....	52
Close Project Files	53
Save the Project File	54
Change Properties for a Project.....	55
4.6 The Run Menu.....	56
Check Syntax of the Model	57
Solve the Model.....	58
Parse the Model Into Memory	59
Solve the Model Currently in Memory	59
Generate Solution File	59
Clear the Current Model From Memory	60
Generate Mapping	60
Generate Input File	60
4.7 The View Menu.....	61
View the Solution Files.....	62
View Other Files.....	63

View Solution Values	64
View the Model Statistics	65
View Model Definitions Window	66
View Message Window	70
4.8 The Graph Menu.....	71
Graph of the Matrix	71
Graph of the Objective Function	73
4.9 The Options Menu	74
Change MPL Environment Options	75
Change MPL Language Options	77
Database Options	80
Change Solution File Options.....	82
Change Generate File Options.....	84
Change General Solver Options	85
Change CPLEX Simplex Options	87
Change CPLEX Preprocessing Options	90
Change CPLEX Log File Options	93
Change CPLEX Limit Options	95
Change CPLEX MIP Strategy Options	98
Change CPLEX MIP Strategy2 Options	100
Change CPLEX MIP Cuts Options	103
Change CPLEX MIP Tolerance Options.....	103
Change CPLEX Barrier Options	108
Change CPLEX Network Options.....	111
Change XPRESS Simplex Options	113
Change XPRESS Preprocessing Options	116
Change XPRESS Log File Options	118
Change XPRESS Limit Options	120
Change XPRESS Tolerance Options.....	122
Change XPRESS MIP Strategy Options	124
Change XPRESS MIP Cuts Options	126
Change Barrier Options for XPRESS.....	128
Change Solver Parameter Options.....	130
Solver Options List Dialog Box.....	131
Setup Solvers for the Run Menu.....	132
Change Setup Options for Solvers.....	133
4.10 The Window Menu.....	135
Tile and Cascade Windows	135
Arrange Minimized Icons	135
Close All Windows.....	135
4.11 The Help Menu.....	135
Using the MPL Help System	136
About MPL for Windows	139

CHAPTER 5 LANGUAGE OVERVIEW 143

5.1 Structure of the MPL Model File.....145
 The Problem Title147
 The Definition Part147
 The Model Part148
5.2 Basic Input Elements149
 Numbers.....149
 Names149
 Delimiters150
 White Space150
 Inserting Comments150
 Include Files.....151
 Conditional Directives151
 Option Settings152

CHAPTER 6 DEFINING SETS AND INDEXES 155

6.1 Numeric Indexes156
6.2 Named Indexes157
6.3 Alias Indexes158
6.4 Circular Indexes.....158
6.5 Subsets of Indexes159
6.6 Function Indexes.....160
6.7 Set Operations on Indexes161
6.8 Multi-dimensional Index Sets162
6.9 Reading Index From External File.....164
6.10 Import Index from Excel Spreadsheet165
6.11 Import Index from Database166

CHAPTER 7 DATA FOR THE MODEL 167

7.1 Data Constants.....168
7.2 Data Vectors170
7.3 Sparse Data Vectors171
7.4 Data Arithmetic172
7.5 Reading Data From External Files.....172
7.6 Import Data from Excel Spreadsheet.....174
7.7 Import Data from Database176

CHAPTER 8 FORMULATING THE MODEL 177

8.1 Declaring Decision Variables178
 Variable Names178
 Integer Variables.....179
 Initial Values for Variables.....179

Where Conditions on Variables.....	180
Export Variable Values to Data Files	180
Export Variable Values to Excel Spreadsheet	181
Export Variable Values to Database.....	182
8.2 Defining Macros.....	183
8.3 The Objective Function	184
8.4 Specifying Constraints.....	185
Plain Constraints.....	185
Vector Constraints	187
Where Conditions on Constraints	188
Export Constraint Values to Data File.....	189
Export Constraint Values to Excel Spreadsheet	189
Export Constraint Values to Database.....	190
8.5 Bounds on Variables.....	191
Free Variables.....	192
Semi-Continuous Variables	192
8.6 Integer and Binary Variables	193
Special Ordered Sets of Variables	193
CHAPTER 9 BUILDING FORMULAS	195
9.1 Coefficients for Variables.....	196
9.2 Using Arithmetic Functions.....	197
9.3 Using Variables in Formulas	198
9.4 Formula Terms.....	200
9.5 IF/IIF Conditions on Formula Terms.....	202
9.6 Where Conditions on Formula Terms	202
9.7 Referring to Indexes in Formulas	203
9.8 Using Parentheses.....	204
9.9 Summing Vectors Over Index Values	205
9.10 Using Macros.....	206
9.11 Abort If Conditions.....	207
CHAPTER 10 ADVANCED INDEXING TECHNIQUES	209
10.1 Set Membership with the IN Operator.....	210
10.2 Set Domain Index with the Dot Operator	212
10.3 Set subsets with the OVER operator	213
10.4 Subscript Arithmetic with Conditional Indexes.....	214
10.5 Direct Assignment of Subscripts	215
CHAPTER 11 DATABASE CONNECTION.....	217
11.1 Import Indexes from Database.....	219
11.2 Import Data Vectors from Database	221
11.3 Export Variable Values to Database.....	223
11.4 Export Constraint Values to Database.....	226

Tutorial Overview	231
SESSION 1: Running MPL on a Sample Model.....	233
1.1 Your First MPL Session	234
1.2 Using the Help System in MPL	241
SESSION 2: Formulating a Simple Product-Mix Model	243
2.1 Problem Description: A Simple Product-Mix Model	245
2.2 Formulating the Model	246
Identify the Decision Variables	246
Identify the Objective Function	246
Identify the Constraints.....	247
Summing up the Formulation	247
2.3 Solving the Model in MPL	248
Step 1: Start MPL and Create a New Model File	248
Step 2: Enter the Model Formulation for the Bakery Model	248
Step 3: Check the Syntax of the Model	249
Step 4: Solve the Model.....	251
Step 5: View and Analyze the Solution	252
SESSION 3: Introducing Vectors and Indexes in MPL Models.....	255
3.1 New Concepts in this Session.....	256
Indexes as the Domains of the Model.....	256
Data, Variable, and Constraint Vectors	256
Data Constants	257
Using Summations over Vectors	257
3.2 Problem Description: A Product-Mix Model with Three Variables.....	258
3.3 Formulation of the Model in MPL.....	259
3.4 Enter the Model in MPL Step-by-Step	260
3.5 Solve the Model and Analyze the Solution	264
Solve the Model.....	264
View and Analyze the Solution	265
SESSION 4: Planning Model with Multiple Time Periods	269
4.1 New Concepts in this Session.....	270
Period Indexes	270
Sales and Inventory Variables	270
Inventory Balance Constraints.....	270
Initial and Ending Inventory	271

4.2	Problem Description: A Multi-Period Production Planning Model	272
4.3	Formulation of the Model in MPL.....	273
4.4	Enter New Elements to the Model Step-by-Step	274
4.5	Solve the Model and Analyze the Solution	279
SESSION 5: Production Planning Model with Multiple Plants		283
5.1	New Concepts in this Session.....	284
	Plants and other Location Indexes	284
	External Data Files	284
5.2	Problem Description: Planning Model with Multiple Plants.....	285
5.3	Formulation of the Model in MPL.....	286
5.4	Enter New Elements to the Model Step-by-Step	287
5.5	Solve the Model and Analyze the Solution	292
SESSION 6: Allow Shipments Between the Plants		295
6.1	New Concepts in this Session.....	296
	Transportation Models.....	296
	Transshipment Models.....	296
	Alias Indexes	296
	Using Where Conditions on Vector Variables	296
	Plant Balance Constraints	297
6.2	Problem Description: Additions to Allow Shipments Between Plants.....	298
6.3	Formulation of the Model in MPL.....	300
6.4	Enter New Elements to the Model Step-by-Step	301
6.5	Solve the Model and Analyze the Solution	306
SESSION 7: Formulating Models With Sparse Data in MPL		309
7.1	New Concepts in this Session.....	310
	Equipment Indexes	310
	Using the IN Operator	310
	Index Files	311
	Sparse Data Files	311
7.2	Problem Description: A Planning Model with Multiple Machines	313
7.3	Formulation of the Model in MPL.....	314
7.4	Enter New Elements to the Model Step-by-Step	316
7.5	Solve the Model and Analyze the Solution	320

APPENDICES **323**

APPENDIX A: CHARACTER SET..... 325

APPENDIX B: ERROR MESSAGES 327

Format Errors in the Formulation	327
Errors for Nonlinear Parser.....	340
Errors Using Include Files	341
Errors Using Conditional Directives	341
Errors with Data Files	342
Errors Reading MPS Files	342
Errors with Problem Size and Memory	342
Errors Using Database Connection.....	344

INDEX **347**

PART I

GETTING STARTED

Chapter 1: Introduction

Chapter 2: Getting Started with MPL

Part I Getting Started



CHAPTER 1

INTRODUCTION

Optimization is today one of the most important tools in implementing and planning efficient operations and increasing competitive advantage. Organizations need to make intelligent decisions to obtain optimal use of their available resources, such as manpower, equipment, raw materials and capital. The discipline of optimization, through the use of advanced mathematics and computer science techniques, can be used to assist organizations with solving their complex business problems in areas such as manufacturing, distribution, finance, and scheduling. Typically, these optimization problems contain hundreds, thousands, or even millions of interconnected variables and require an advanced set of software tools to solve.

Today, the field of optimization entails highly advanced software applications that integrate sophisticated mathematical algorithms and modeling techniques with intelligent software programming and data processing capabilities.

Optimization projects begin with the development of a mathematical model that defines the business problem. Individual business decisions are represented as “variables,” and the connections between them are represented by a series of mathematical equations termed “constraints”. The “objective” represents the goal of the business problem, for example, to maximize profitability or lower costs. Identifying the variables, the constraints and the objective is known as the “modeling” process and is an essential task for every optimization project. After the model has been formulated, it is then solved, using an optimization solver, which, at its core, has highly sophisticated algorithms adept at intelligently sorting through huge amounts of data and analyzing possible approaches to come up with an optimal solution.

1.1 What Is MPL?

Maximal Software is the developer of **MPL** (**M**athematical **P**rogramming **L**anguage) an advanced modeling system that allows the model developer to formulate complicated optimization models in a clear, concise, and efficient way. Models developed in **MPL** can then be solved with any of the multiple commercial optimizers available on the market today.

MPL includes an algebraic modeling language that allows the model developer to create optimization models using algebraic equations. The model is used as a basis to generate a mathematical matrix that can be relayed directly into the optimization solver. This is all done in the background so that the model developer only needs to focus on formulating the model. Algebraic modeling languages, such as **MPL**, have proven themselves over the years to be the most efficient method of developing and maintaining optimization models because they are easier to learn, quicker to formulate and require less programming.

MPL offers a feature-rich model development environment that takes full advantage of the graphical user interface in Microsoft Windows, making **MPL** a valuable tool for developing optimization models. **MPL** can import data directly from databases or spreadsheets. Once the model has been solved, **MPL** also has the ability to export the solution back into the database. **MPL** models can be embedded into other Windows applications, including databases and spreadsheets, which makes **MPL** ideal for creating end-user applications.


The main purpose of a modeling language is to retrieve data from a structured data source, such as a database, and generate a matrix that the optimization solver can handle. For large optimization models, this matrix generation requires a modeling language with highly advanced capabilities, such as sparse indexing and database management, as well as high scalability and speed. Many details need to be taken into account when choosing a modeling language for optimization projects:

- Model Development Environment
- Robustness and Flexibility of the Modeling Language
- Indexing and Data Management
- Scalability and Speed
- Database Connections
- Connection to Solvers
- Deployment into Applications

MPL was designed to support multiple platforms. **MPL for Windows** is the most popular platform but an OSF Motif version is also available for various UNIX flavors including HP 9000, IBM RS-6000, Sun Sparc, Silicon Graphics and Linux. **MPL** models are portable so a model created for one platform can always be read on any other supported platform.

This release of **MPL**, offers the highest performance of modeling languages on the market today. Since we are constantly working on new releases of **MPL**, please contact Maximal Software for updated information.

1.2 Key Features of MPL

- **Graphical User Interface.** MPL is an integrated model development system that offers full support for all standard Windows features, including dialog boxes, mouse support, pull-down menus, graphics, toolbar, and on-line help. Furthermore, the new version of MPL, supports advanced Windows graphical features such as tree windows, long filenames for models, illustrative icons for each type of window, and context sensitive help.
- **Direct link to Windows DLL solvers.** MPL can set up the matrix and then send it to the solver directly through memory. Solvers that are supported in MPL as Windows DLL include, CPLEX, XPRESS, OSL, XA, CONOPT, LSGRG2, and FortMP. MPL can also handle legacy DOS solvers through a DOS window.
- **Database Connection.** MPL can import both indexes and data directly from a database. After the model has been solved MPL can also export the solution back to the database. Furthermore, MPL can easily be called directly from other Windows applications, including databases. Supported databases include Access, Paradox, FoxPro, Dbase and any ODBC compatible database.
- **Manage models through projects.** If you are using MPL to work on multiple models, that use different files and option settings, you can use projects to manage the models. Projects are used to store information about items such as, open model files and windows, the default working directory, and current option settings.
- **Model Definitions window.** MPL allows you to view defined items from the model formulation in an easy to browse tree window. You can expand and collapse each branch to show only the elements you are interested in. You can display the contents for each item, such as elements for an index or solution values for a variable, simply by selecting it in the tree.
- **Message window.** While MPL is running it can send various progress information to a message window. What is displayed is selected by the user and can include status window messages, MPL input lines, performance statistics, warning messages, SQL statements, and iteration log information from the solver.
- **Context sensitive help.** MPL Supports Windows context sensitive help for dialog boxes. To display the help, simply select the question mark button  in the upper right corner of the dialog box and then click on the item you want help for. A small window will popup with a short explanation of the item you selected.
- **Multiple input formats.** In addition to the MPL modeling language, MPL can read multiple other input formats, including MPS files and native input formats for solvers such as CPLEX. MPL will automatically detect the format, when reading the input file, and switch to the correct language. MPL can also be used to generate several different solver input formats. This ensures that models created in MPL can be solved with nearly all industrial strength optimization packages available on the market today.

Part I Getting Started

- **Helpful error messages.** **MPL** makes it easy to correct mistakes in your formulation. An error window pops up containing the erroneous line in the model file with the error is described in plain English. After you have read the message, the program automatically locates the line in the model file and moves the cursor to it.
- **Meaningful names.** You can use meaningful names of any length for variables and constraints, so the formulation is easy to read and understand. You can also, for additional clarity, use explanations and general comments throughout the model file.
- **Free format constraints.** With **MPL** you can write variables and constants on both sides of the constraints. This means that you do not have to convert the constraints to a standard format before entering them.
- **Arithmetic in the input.** You can use fractions, products, percentages, and mathematical functions in the model file. This flexibility not only allows clearer formulation, but also automatically results in the highest possible accuracy. Full use of parentheses is also allowed.
- **Separation of data from the model.** With **MPL** you can read in data from external data files in both sparse and dense data formats. The data files can be created either using the text editor in **MPL** or be exported from other programs such as spreadsheets and databases. For example, to read in a data file containing a price list you would simply enter:

```
price[product] := datafile(price.dat)
```

- **Data entered at run-time.** Named data constants can be given a value interactively at run-time. You are prompted for the value when the model file is read. This feature can be very useful when you want to run the same model several times with different data values.
- **Summation over vector variables.** In **MPL** you can work with vector variables of up to eight dimensions concisely and effectively. For example, to add the production cost for all months and products, you simply type:

```
SUM(month,product: ProdCost * Production)
```

- **Expansion of structured constraints.** Models generally have many similarly structured constraints. In **MPL** you can enter these constraints in a single line that is then expanded to a list of constraints. For example, to enter 12 constraints, one for each month, you would type:

```
InventoryBalance[month=Jan..Dec] :  
Inventory = Inventory[month-1] + Production - Sales ;
```

- **Macro definitions.** Macro definitions in **MPL** allow you to define a part of a constraint or formula that is frequently used as a macro. You can then refer to it in the model by entering just the macro name. In some cases, macros can be used to eliminate unnecessary equality constraints and thereby reduce the size of the problem for the solver.
- **Include files.** **MPL** allows you to include external files to the model formulation, using a simple include command. This can make the model formulation more modular and easier to read. Include files can also be a great advantage when you want to solve a number of closely related models containing common data or model statements.

1.3 How to Contact Maximal Software

If you have any questions regarding the use of **MPL**, please feel free to contact us for assistance. We will do our best to help you solve your problems.

Maximal Software, Inc.

2111 Wilson Boulevard, Suite 700
Arlington, VA 22201
U.S.A.

Tel: (703) 522-7900

Fax: (703) 522-7902

Email: info@maximalsoftware.com

Web: www.maximalsoftware.com

Maximal Software, Ltd.

One Oxford Road
Uxbridge, Middlesex, UB9 4DA
United Kingdom

Tel: +44 (0)1895 819 344

Fax: +44 (0)1895 819 345

Email: info@maximalsoftware.co.uk

Web: www.maximalsoftware.co.uk

Maximal Software, Ehf.

Nordurasi 4
IS-110 Reykjavik
ICELAND

Tel: +354 587-7700

Fax: +354 588-9728

MPL is continually evolving in order to bring you the most current and advanced techniques available. Your input is a helpful component in shaping our focus and direction and is greatly appreciated.

Part I Getting Started



CHAPTER 2

GETTING STARTED WITH MPL

This chapter tells you what is needed to begin using **MPL**. The chapter contains these sections:

- *System Requirements* shows you the hardware and operating system requirements for using the **MPL** software.
- *Setting Up Solvers* gives information about changing which solvers are available in the *Run* menu in **MPL** and all solvers that are supported by the current version of **MPL**.
- *Installing MPL for Windows* gives information about installing **MPL** and what files are contained on your **MPL** distribution diskette.
- *Using the MPL Environment* gives you a short overview on how to run the program.

2.1 System Requirements

The minimal system requirements for running *MPL for Windows* are as follows:

- IBM PC compatible with 486 or Pentium processor
- 16 Mb of memory
- 3 Mb free hard disk space
- Microsoft Windows 95/98/Me or Windows NT/2000/XP

2.2 Installing MPL for Windows

Before you install **MPL**, to protect your software from loss, please copy all the files on the distribution disk onto a backup and put in a safe place.

To install *MPL for Windows* place the first **MPL** installation disk in drive A:. From *Start* menu choose the *Run* command. Type *a:\setup* and click *OK*. The setup program will ask number of simple questions such as where to install the software and then copy the files to the hard disk on your machine. It will then create entries for **MPL** in the *Start* menu.

The files on the distribution diskette for **MPL** are copied to the hard disk as follows:

Mplwin4 directory:

Mplwin42.exe	Contains the MPL for Windows application.
Mplwin42.hlp	On-line help file for MPL.
Mplwin42.cnt	Contents file for the on-line help.
Mplwhat.hlp	Context sensitive help file for MPL.
*.mpl	Contains sample input files, with full formulations of various LP problems. These files tell you how to use many of the MPL features.
*.dat	Contains the data files that accompany the sample model files.
Mplwin42.ini	Initialization file for mplwin that stores the current setup.
*.opt	Option files for some of the solvers supported by MPL.
Tutorial	Directory containing sample models for the on-line tutorial.

Windows directory:

Mpl*.pif	PIF files for running solvers in a DOS window.
Mpl*.bat	Batch files for running solvers in a DOS window.

2.3 Setting up Solvers for MPL

When **MPL** is run for the first time, after installing the software, it will automatically search the hard disk for supported solvers. If the solver was not installed with **MPL** and you do not have a solver already on the machine, you need to install one before **MPL** can be used to solve models.

After you have installed the solver, you can change which solvers are available in the *Run* menu by choosing *Solver Menu* from the *Options* menu. This will display the *Solver Menu Setup* dialog box. The list box shows the solvers that are supported by the current version of **MPL**.



Figure 2.1: Solver Menu Setup Dialog Box

To let **MPL** search your hard disk for supported solvers, press the *Scan* button. This option can be especially useful when you are not sure where on the hard disk solvers have been installed and you want **MPL** to locate them automatically and set them up.

You can select manually which solvers should be in the *Run* menu. Double-click on a solver name to either add or remove it from the menu. Those solvers that are currently in the menu have the word 'Menu' listed in the second column.

If you need to change some of the setup options for a solver, such as the filename or the location, select the solver in the list box and then press the *Edit* button. This will display the *Solver Setup Options* dialog box. For further information on how to setup solvers for **MPL** refer to the sections on *Setup Solvers for the Run Menu* and *Change Setup Options for Solvers* in Chapter 4.9 *The Options Menu*.

2.4 Using the MPL Environment

To run the **MPL** program, select *MPL for Windows* from the *Start* menu. *MPL for Windows* is a fully integrated model development environment with multiple windows and dialog boxes, pull-down menus, toolbar, model editor, and an on-screen output viewer.

To start working on your model choose from the *File* menu either *New* to create new model or *Open* to edit an existing model. To save your model use either *Save* or *Save As* from the *File* menu.

After you have worked on your model in the editor choose *Solve <solvername>* from the *Run* menu to solve the model.

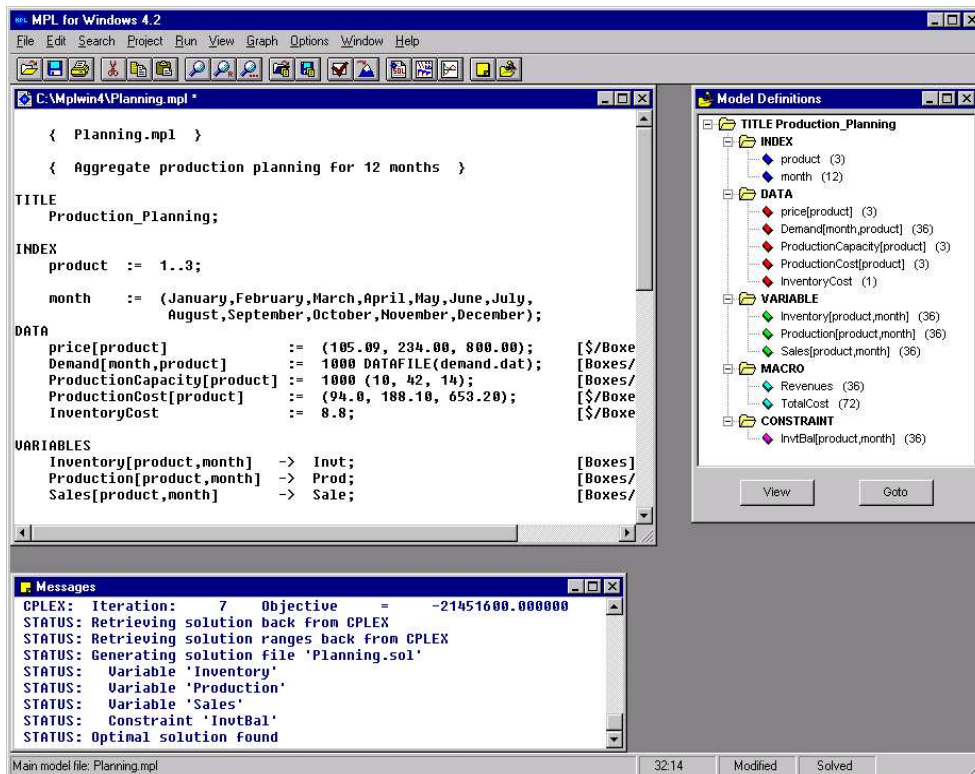


Figure 2.2: MPL Model Development Environment

While optimizing, **MPL** displays a status window that gives you information on the solution progress. When the model has been solved, the solution is written to a solution file. You can use options from the *View* menu to display various parts of the solution. You can also use the *Graph* menu to display a graph of the matrix or the objective function.

PART II

USING THE MPL

MODELING SYSTEM

Chapter 3: Running MPL

Chapter 4: Description of Menus

Part II Using the MPL Modeling System



CHAPTER 3

RUNNING MPL

The *MPL Modeling System* is a state-of-the-art optimization modeling software. Through the use of advanced graphical user interface features, **MPL** creates a flexible working environment that enables the model developer to be more efficient and productive. **MPL** provides in a single system all the essential components needed to formulate the model, gather and maintain the data, optimize the model, and then analyze the results.

The model developer uses the built-in model editor to formulate the model in **MPL** and then selects the optimizer directly from the menus to solve the model. The solution results are automatically retrieved from the solver and displayed, providing the user with instant feedback. Each item defined in the model is also displayed in a tree window allowing the model developer to browse through them easily.

The *MPL Modeling System* connects to solvers dynamically through memory at run-time. This gives **MPL** the capability to integrate the solver completely into the modeling environment, resulting in the matrix being transferred between the modeling system and the solver directly through memory. As no files are involved, this seamless connection is both considerably faster and more robust than the traditional use of files in other modeling systems. In the event it is necessary to change any algorithmic options, **MPL** provides easy-to-use option dialog boxes for each solver.

This chapter describes the process of running **MPL** in detail. It contains these sections:

- *How to Start MPL* demonstrates how to run **MPL** on your computer.
- *The Main Window* explains various parts of the main window in **MPL**.
- *Using the MPL Environment* gives you step by step explanations on how to use **MPL**.

3.1 How to Start MPL

There are several ways you can start **MPL** in Windows. The two most common are either through the *Start* menu or by creating a *shortcut* for it.

Running MPL from the Start Menu

The installation program for **MPL** created an entry in the *Start* menu under *Programs / MPL for Windows*. To access it click the *Start* button from the task bar that appears along the bottom of the Windows screen. The *MPL for Windows* start menu entry is shown here in Figure 3.1:

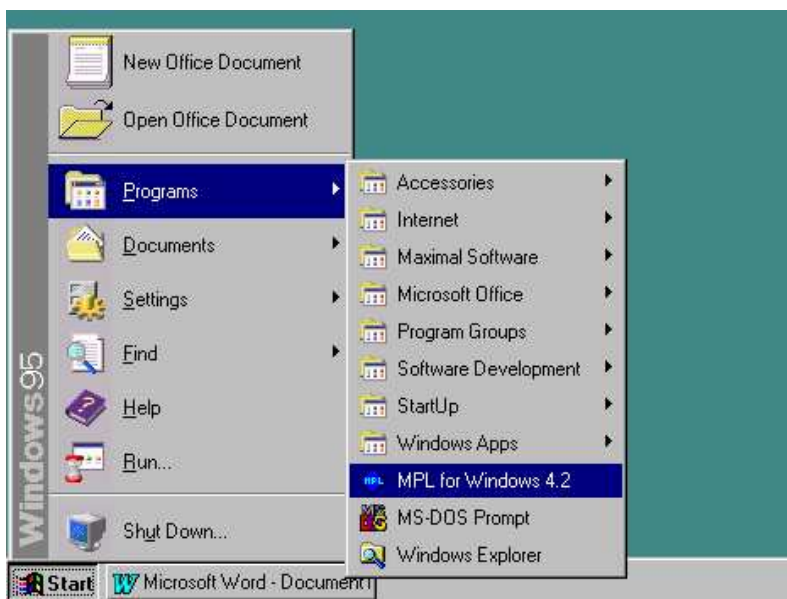


Figure 3.1: MPL for Windows in the Start Menu

Create a Shortcut for MPL

You can create a *shortcut* for **MPL** by clicking on the right mouse button anywhere on the desktop. You will be presented with a pop-up menu where you choose *New* and then *Shortcut*. This will start the *Create Shortcut Wizard*. In the input box enter the full path for the **MPL** program. If you do not know the path you can press the *Browse* button to locate **MPL** in your system. After entering the path, press *Next* to continue. In the next window enter the name of the shortcut and then press *Finish*. The new shortcut will now be created on the desktop.

You can also set up **MPL** so it will automatically open a specific model file when starting. There is an example of this in Figure 3.1 where you can see the **MPL** icon with the caption '*Planning.mpl*'. To see how this is set up, create a shortcut for **MPL** on the desktop and then use the right mouse button to choose properties for it. This will bring up the *Properties* dialog box. Select the *Shortcut* tab and in the *Target* input box enter the following:

```
c:\mplwin4\mplwin42.exe planning.mpl
```

By giving the name of the model file '*planning.mpl*' as an argument after the program name on the command line, **MPL** will automatically open the file when it is started. To create another icon that opens a different file, first duplicate the icon by dragging it with the right mouse button and select *Copy Here*. Then you can change the name of the model file in the command line by going again into the *Properties* dialog box.

Embedding MPL Models in Applications

MPL was not only designed to help the model developer to create optimization models, but also to make it easy to deploy the model by embedding it into business applications. With the introduction of the **OptiMax 2000 Component Library**, **MPL** models can now be easily embedded into end-user applications such as *Excel* and *Access* from *Microsoft Office*, using programming languages such as *Visual Basic* and *C++*. This allows model developers whose expertise lies in developing models, to effectively work with and deliver models to the IT professionals who can then focus on working in databases and building graphical user interfaces and end-user applications. Please contact Maximal for more details on the **OptiMax 2000 Component Library**.

Running MPL in Run-time Mode

MPL can be run directly from other windows applications in run-time mode. This will let **MPL** solve a model from the other application without entering the integrated modeling system. **MPL** is run in run-time mode by entering the command *SOLVE* to the command line before the name of the model.

```
c:\mplwin4\mplwin42.exe SOLVE capri.mpl
```

This will direct **MPL** to read the model file that follows and solve the model directly. You can also ask **MPL** to generate an input file, such as *MPS*, in run-time mode, by entering the command *GENERATE* to the command line followed by the *MPS* filename.

```
c:\mplwin4\mplwin42.exe GENERATE capri.mps
```

This will direct **MPL** to read the model file, with the filename that follows, using the extension '*.mpl*' and then generate the input file you requested. **MPL** will determine which type of input file to generate from the file extension used.

Calling MPL from Other Applications

Many end-user applications for Windows, such as databases and spreadsheets, have a macro language that lets you execute other programs from a command line. This allows you to use the application as a front-end that handles the data entry and management tasks for your project and then call **MPL** directly from the application to solve the model. Then, using the database features of **MPL**, you can create an application around your model where the end-user never has to know how to use **MPL**.

The way you can call **MPL** from other applications depends on which software you are using. The most common applications that are used include Visual Basic, C/C++, and databases such as MS Access. In most cases this is accomplished by making a Win32 call to *CreateProcess*. Other programs will possibly have functions like *WinExec* or *Shell*. In many cases, you will want your calling application to wait until **MPL** and the solver are finished before continuing. Here is an example of a *WinExecWait* function that will first call *CreateProcess* and then use the Win32 function *WaitForSingleObject*.

```
int WinExecWait(LPSTR CommandLine)
{
    UINT Result;
    DWORD dwExitCode;
    STARTUPINFO StartupInfo = {0};
    PROCESS_INFORMATION ProcessInfo;

    StartupInfo.cb = sizeof(STARTUPINFO);
    Result = CreateProcess(NULL, CommandLine, NULL, NULL, FALSE, 0, NULL, NULL,
                          &StartupInfo, &ProcessInfo);

    if (!Result)
        return WE_SOMEERROR;
    else {
        CloseHandle(ProcessInfo.hThread);
        if (WaitForSingleObject(ProcessInfo.hProcess, INFINITE) != WAIT_FAILED) {
            GetExitCodeProcess(ProcessInfo.hProcess, &dwExitCode);
        }
        CloseHandle(ProcessInfo.hProcess);
        return WE_RANOK;
    }
}
```

This *WinExecWait* function can now be called using a command line that runs **MPL** in the run-time mode. For example:

```
WinExecWait("c:\\mplwin4\\mplwin42.exe SOLVE capri.mpl");
```

Visual Basic supports a *Shell* function that allows you to run external programs, such as **MPL**, but it does not wait for the program to finish. If you need a function that waits until **MPL** finishes, Visual Basic supports *Win32* function calls and, therefore, will be very similar to the above example. Please contact Maximal Software if you need sample copy of *WinExecWait* for Visual Basic.

3.2 The MPL Main Window

When you start **MPL** in stand-alone mode the *Main Window* appears. The Main Window consists of the title bar at the top, the main menu, the toolbar, the work area, and the status line at the bottom.

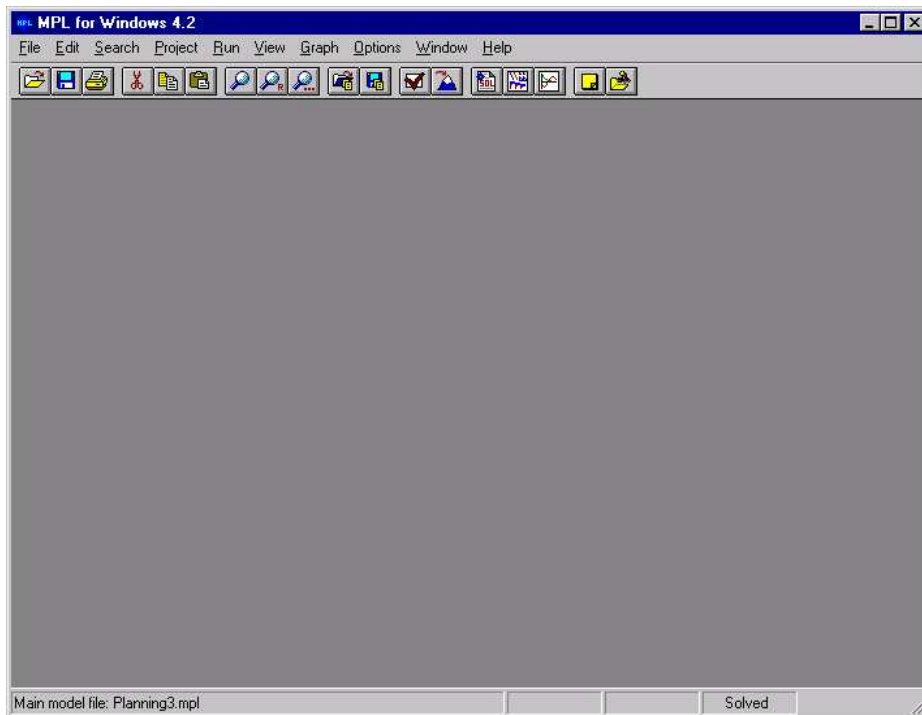


Figure 3.2: The MPL Main Window

The Main Menu

The main menu in **MPL** offers the following menus and functions:

File	- Open and save model files.
Edit	- Undo, cut, copy, and paste commands.
Search	- Search and replace, goto commands.
Run	- Run solver, check syntax, and generate input
Project	- Open, save, and close projects
View	- View the output on screen
Graph	- Display graph of matrix and objective
Options	- Change default options for MPL
Window	- Handling of multiple windows
Help	- On-line help for MPL

The Pull-down Menus

Here is a quick overview of all the pull-down menus that are available in **MPL**. For more detailed information please refer the *Chapter 4. Description of Menus*.

File menu

New	- Create a new MPL model file
Open...	- Open an existing model file
Close	- Close the current editor window
Save	- Save the current model file to disk
Save As...	- Save model file under a new name
Save Selection...	- Save selected text to disk
Insert File	- Insert file from disk into current model
Print	- Print the current model file
Exit	- Quit MPL

Edit menu

Undo	- Undo last change to the model file
Cut	- Move selected text to the clipboard
Copy	- Copy selected text to the clipboard
Paste	- Insert text from the clipboard
Delete	- Delete text from the model file
Select All	- Select all text in the model file

Search menu

Find	- Search for text in the model file
Replace	- Replace text in the model file
Next	- Search for the next occurrence of text
Goto Line	- Goto line in the model file

Project menu

New Project	- Create a new project file
Open Project	- Open an existing project file
Close Project	- Close the current project
Save Project	- Save the current project to disk
Save as Project	- Save project under a new name
Properties	- Changing properties for the current project

Run menu

Check Syntax	- Check syntax of the current MPL model file
Solve <solvername>	- Solve the model using the specified solver
Parse Model	- Parse the model file into memory
Solve Current	- Solve the model again without parsing
Generate Solution	- Generate solution file for last solver run
Clear Model	- Clear the current model from memory
Generate Mapping	- Generate mapping file for the solution
Generate File <solver>	- Generate input file for external solver

View menu

- Files <solution file> - View the MPL solution file generated
- Files <output file> - View the output file from the solver
- Files <input file> - View the input file for the solver
- Files <log file> - View the log file for the solver run
- Files Other Files - View any other file on the disk
- Values/Reduced Cost - View variable values and reduced costs
- Slack/Shadow Prices - View constraint slacks and shadow prices
- Range Objective - View ranges for the objective function
- Range RHS - View ranges for the right-hand-side
- Model Statistics - View the statistics of the current model
- Model Definitions - Open the model definitions window
- Message Window - Open the message window

Graph menu

- Matrix - Display graph of the matrix
- Objective Function - Display graph of the objective function

Options menu

- Environment... - Change options for the MPL environment
- MPL Language... - Change options for the MPL language
- Database... - Change options for the database connection
- Solution File... - Change contents of the solution file
- Generate File... - Change options for generated files
- General Solver... - Change general options for solvers
- CPLEX parameters... - Change option parameters for CPLEX
- XPRESS parameters... - Change option parameters for XPRESS
- Solver Parameters... - Change parameters for available solvers
- Solver Options List... - Change various options for supported solvers
- Solver Menu... - Sets up the menu of available solvers

Window menu



















- Tile - Divides the screen equally between windows
- Cascade - Arranges windows on top of each other
- Arrange Icons - Arranges iconed windows at the bottom
- Close All - Closes all windows

Help menu

- Contents - Display the help system contents window
- Search for Help on - Search for a specific help topic
- About MPL - Display the About MPL for Windows dialog.

The Toolbar

The *Toolbar* in **MPL** is located at the top of the window directly below the main menu. It allows you to choose a menu command more quickly by simply pressing the button. The toolbar contains many of the most commonly used commands in **MPL**. The following is a list of the available buttons:

Button	Menu	Description
	File Open	Open an existing model file
	File Save	Save the current model file to disk
	File Print	Print the current model file
	Edit Cut	Move the selected text to the clipboard
	Edit Copy	Copy the selected text to the clipboard
	Edit Paste	Insert text from the clipboard
	Search Find	Search for text in the model file
	Search Replace	Replace text in the model file
	Search Next	Search for the next occurrence of text
	Project Open	Open an existing MPL project file
	Project Save	Save the current project file to disk
	Run Check Syntax	Check syntax of the current model file
	Run Solve	Solve the model using the default solver
	View Solution File	View the solution file for the current model
	Graph Matrix	Display graph of the matrix
	Graph Objective	Display graph of the objective function
	View Message Window	Open the message window
	View Model Definitions	Open the model definitions window

The Status Bar

The *Status Bar* in **MPL** is located at the bottom of the main window. It is used to report back to the user various information about status of the current model.

MPL automatically remembers the last model that was run and displays the name of it in the *message area*, to the left of the Status Bar. This allows the user to quickly see the current model file and solve it again, without having to find the window containing the model file again and bring it to the front.



Figure 3.3: The Status Bar for MPL

Directly to the right of the message area are three smaller notification areas where **MPL** reports various information about the status of the model.

The first area is used to show the current line and column number of the insertion point in the model editor window.

The second area is used to report whether the current model file has been modified or if the file is too big to be edited and, therefore, can only be viewed.

The last area is used to report the current state of the model in memory. If *Parsed*, the model has been parsed into memory, but not yet solved. If *Solved*, the model has been solved and the solution is available for viewing.

While the pull-down menus are being used, **MPL** will turn off the standard Status Bar and instead display a short explanation of the currently selected menu item. This allows the user to quickly see, while browsing through the menus, the explanation.

3.3 Using the MPL Environment

The following is a quick overview on how to work in **MPL** and what you can do with each menu.

To start working on your model go to the *File* menu and choose either *New* to create a new model or *Open* to edit an existing model file. When you have finished editing the model you can save it using either *Save* or *Save As* from the *File* menu.

You can use the *Cut* and the *Copy* command from the *Edit* menu to copy block of text to the clipboard and then the *Paste* command to place it elsewhere in your model. If you make a mistake while entering your model you can use the *Undo* command from the *Edit* menu to correct it.

If you need to search for a text string in your model, you can use the *Find* command from the *Search* menu or if you need to replace the text with an another text string use the *Replace* command.

If you are using **MPL** to work on multiple models, that use different files and option settings, it may be beneficial to use *Projects* to manage the models. Projects are used to store information about open model files and windows, the default working directory, and all the current option settings. The *Project* menu is used to create new projects, and then open, save or close different project files.

After you have created your model in the editor choose *Solve <solvername>* from the *Run* menu to optimize the model. While optimizing, **MPL** displays a status window that gives you information on the solution progress. When the model has been solved, the solution is written to a solution file.

After the solver is finished optimizing the problem, you can use options from the *View* menu to display various parts of the solution. You can also use the *Graph* menu to display a graph of the matrix and of the objective function.

The *Options* menu allows you to change various default settings for **MPL**, including which solvers you have, native solver options, contents of the solution file and other preferences.

The *Windows* menu is used to either tile or cascade the currently open windows, arrange at the bottom minimized windows, and select from a list of open windows, the window you want to bring to the front.

Through the *Help* menu you can access the context sensitive on-line help system which has multiple screens containing useful reference information. While you are running **MPL** you can also enter the help system by pressing the *F1* key.

We will now explain the Status Window and the **MPL** Error Message Window that are used while you are reading and solving your models.

Solver Support in MPL

MPL works with the world's fastest and most advanced solver optimization engines, such as CPLEX and XPRESS and many other industrial strength solvers. **MPL** is designed to have an open architecture and is not restricted to only one solver. This enables the model developer to choose the solver that best suits his specific project needs.

A unique feature of **MPL** is that it links to solvers directly through memory. As no files are involved, this seamless connection is considerably faster and more robust than the traditional use of files in other modeling systems. In the event it is necessary to change any of the algorithmic options for the solver, **MPL** provides easy-to-use option dialog boxes.

MPL has extensive solver support and offer advanced features such as:

- Direct link to solvers through memory
- Status window with progress information
- Fast and efficient correction of errors
- Automatic infeasibility finder
- Log information and warnings displayed
- Setting of solver options through dialog boxes

MPL can work with most commercial solvers currently available on the market today, including the following:

Solver	Supported Algorithms
CPLEX	LP, MIP, BAR
XPRESS	LP, MIP, BAR
OSL	LP, MIP, BAR
FortMP	LP, MIP, BAR, QMIP
XA	LP, MIP
OML	LP, MIP
Lindo	LP, MIP
FrontLine	LP, MIP
LPSolve	LP, MIP
PCx	BAR
CONOPT	LP, NLP
LSGRG2	LP, NLP

CPLEX is one of the most advanced and popular optimization solvers on the market today. It offers a complete solution that contains almost every feature that the model developer would need in an optimization solver. *XPRESS*, from Dash Associates, is a world-renowned solver, which strength lies in its ability to solve very large optimization problems especially mixed integer. *OSL* from IBM is also a very strong optimizer that has the ability to solve many different types of models, including, for example, quadratic mixed integer problems.

There are numerous other solvers on the market today that offer different features that are sometimes not supported by the market leaders; *FortMP* for example from OptiRisk Systems offers quadratic MIP and stochastic programming. Sunset Software Technologies offers a reasonably priced, relatively fast solver, *XA*. Another middle-range solver is *OML* from Ketron Management.

Part II Using the MPL Modeling System

Lindo Systems product, *Lindo*, is popular with academic users. *FrontLine* is from FrontLine Systems, the same company that provides solvers for *Microsoft Excel*. *LPSolve* is a free solver with support for mixed integer programming that is downloadable from the web. Argonne National Laboratories' solver *PCx* is also downloadable from the web and has a very good implementation of Newton Barrier.

MPL currently supports two nonlinear solvers: *CONOPT*, a large-scale solver from ARKI Consulting in Denmark that is highly specialized in solving difficult nonlinear models; and the *LSGRG2* solver made by Leon Lasdon at Optimal Methods that is used, for example, by Excel.

Solving Models in MPL

After you have edited your model in the model editor, choose *Solve <solvername>* from the *Run* menu to let **MPL** start optimizing. The *Status Window* appears, and **MPL** starts reading the model file.

The Status Window

The status window describes what the program is doing (reading, solving, writing), and provides statistics on the problem. While **MPL** reads the model file, the status window displays the number of lines read, the number of variables and constraints encountered, and how much memory has been used. While solving, this window shows you the number of iterations, and the value of the objective function. Figure 3.4 shows an example of a status window:

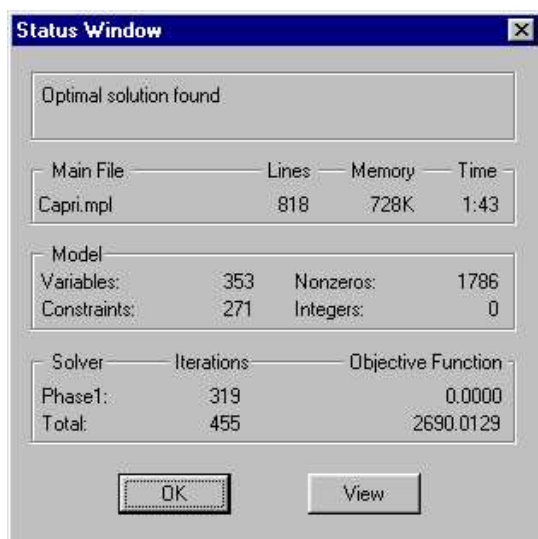


Figure 3.4: The Status Window

The top section in the status window is the *message line*. It tells you what **MPL** is doing at that moment, for example, reading the model formulation, solving the model, and writing the solution

file. The next section contains the name of the main model file, number of lines read so far, how much memory has been used by **MPL**, and time elapsed since the start of the run.

The next section gives you various statistics of the problem, the number of variables and constraints in the model, number of nonzeros, and the number of integer variables.

The last section tells you what the status of the solver is while optimizing the model. It gives you the total number of iterations the solver has done so far. To the right it shows the current value of the objective function and while in Phase 1 the amount of infeasibilities that are left.

For integer problems the last section also reports the number of nodes in the branch and bound tree visited so far, how many improving integer solution have been found, and the best possible integer solution.

The Error Message Window

If **MPL** finds a mistake or an error in the formulation while reading, an *error window* appears. It contains the erroneous line in the model file. Following that line comes a message with a short explanation. *Appendix B: Error Messages* contains a list of all the error messages in **MPL**. Here is an example of a typical error message:

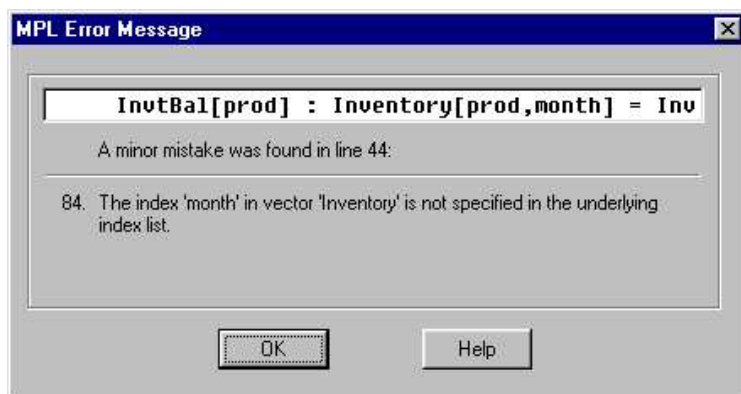


Figure 3.5: The MPL Error Message Window

The above message tells you that the index *month* in the variable vector *Inventory*, was not defined in the underlying constraint *InvBal*. The mistake was located in line 44 of the model file.

Pressing the *OK* button or the *Return* key returns you to the model editor. The cursor is automatically positioned at the location of the error in the model file, with the offending word or character highlighted.

If you need more help on the error message, press the *Help* button. This will give you further explanation of the error, including examples.

The Model Definitions Window

MPL allows you to view defined items from the model formulation in a tree window. Choose *Model Definitions* from the *View* menu, to display the *Model Definitions* window. If you want you can leave this window open while you are working in MPL as it will be updated automatically when you parse in your model.

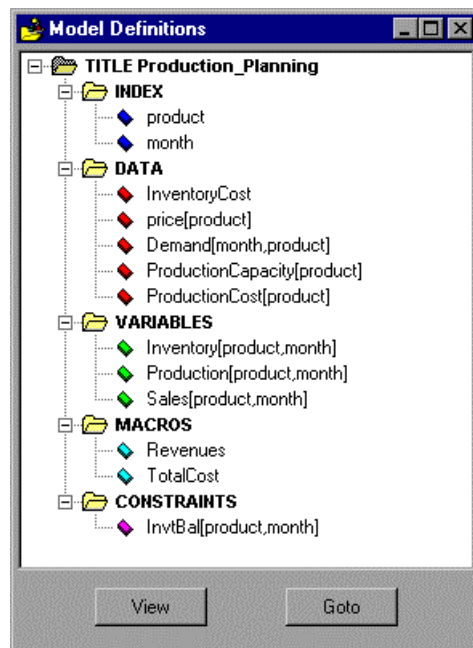


Figure 3.6: The Model Definitions Window

This window will show all the defined items in the model in a hierarchical tree structure. Each branch in the tree corresponds to a section in the model. You can expand and collapse each branch to show only the elements you are interested in.

At the bottom of the window are two buttons. The *View* button allows you to display the contents of each defined item in a separate view window. What is displayed for each item is discussed in the following pages. The *Goto* button will take you directly to the declaration of the selected item in the model file.

MPL has several options in the *MPL Environment Options* dialog box that allow you to control how the Model Definitions window behaves. You can specify whether each defined item is shown with the number of elements it contains. You can also specify whether each branch is expanded or collapsed when the tree is initially created. Finally you can set whether double-clicking on a defined item will work as the *Goto* button or the *View* button.

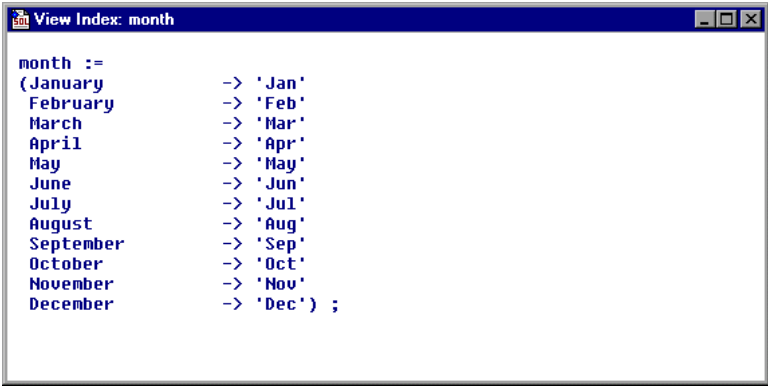
The INDEX Branch

All defined indexes for the current model are listed in the *INDEX* branch. If there are multiple *INDEX* sections in the model they will be shown as separate branches in the tree.

To obtain more information about an index entry, select the index name in the tree and press the *View* button. This will open a view window containing the declaration and contents for the selected index entry. What is shown for each index depends on how it was declared in the model.

- For *numeric indexes* the lower and upper range values are shown.
- For *named indexes* the name of each index element is shown along with the shorter abbreviated name which is used to generate variable and constraint names.
- For *multi-dimensional indexes* all the index elements are shown along with the shorter abbreviated name similar to named indexes.

For example to view the named index *month* select the name in the *INDEX* branch and press the *View* button. This will display the view window shown here below:



```

month :=
(January      -> 'Jan'
 February     -> 'Feb'
 March        -> 'Mar'
 April        -> 'Apr'
 May          -> 'May'
 June         -> 'Jun'
 July         -> 'Jul'
 August       -> 'Aug'
 September    -> 'Sep'
 October      -> 'Oct'
 November     -> 'Nov'
 December     -> 'Dec') ;

```

Figure 3.7: View Named Index Elements

The DATA Branch

All defined data vectors and data constants for the current model are listed in the *DATA* branch. If there are multiple *DATA* sections in the model they will be shown as separate branches in the tree.

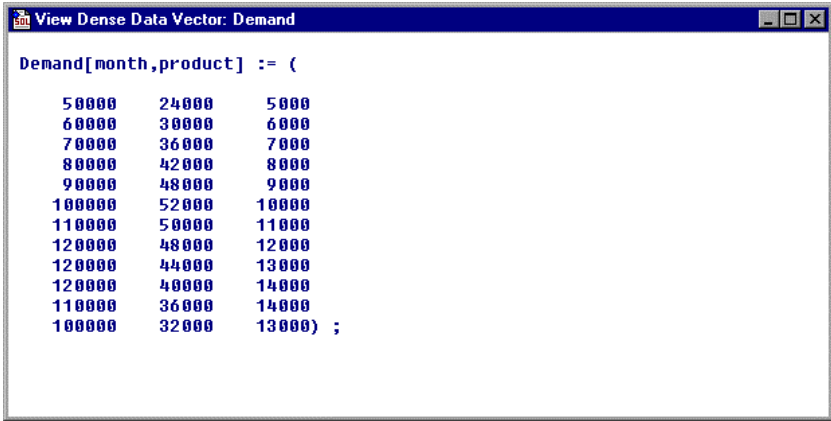
To obtain more information about a data entry, select the data name in the tree and press the *View* button. This will open a view window containing the declaration and contents for the selected data entry.

Part II Using the MPL Modeling System

What is shown for each data entry depends on how it was declared in the model.

- For *data constants* the data value is shown.
- For *dense data vectors* the contents of the data vector is shown as list of numbers organized into rows and columns according to the indexes the data vector was defined over.
- For *sparse data vectors* the contents of the data vector is shown in a table with one line for each element in the vector.

For example to view the dense data vector *Demand* select the name in the *DATA* branch and press the *View* button. This will display the view window shown here below:



The screenshot shows a window titled "View Dense Data Vector: Demand" with a blue title bar. The content of the window is a table of numerical values representing the demand vector. The table is organized into three columns and thirteen rows. The values are as follows:

Demand[month,product] := (
50000	24000	5000
60000	30000	6000
70000	36000	7000
80000	42000	8000
90000	48000	9000
100000	52000	10000
110000	50000	11000
120000	48000	12000
120000	44000	13000
120000	40000	14000
110000	36000	14000
100000	32000	13000

Figure 3.8: View Dense Data Vector Elements

The VARIABLES Branch

All defined variables for the current model are listed in the *VARIABLES* branch. If there are multiple *VARIABLES* sections in the model they will be shown as separate branches in the tree.

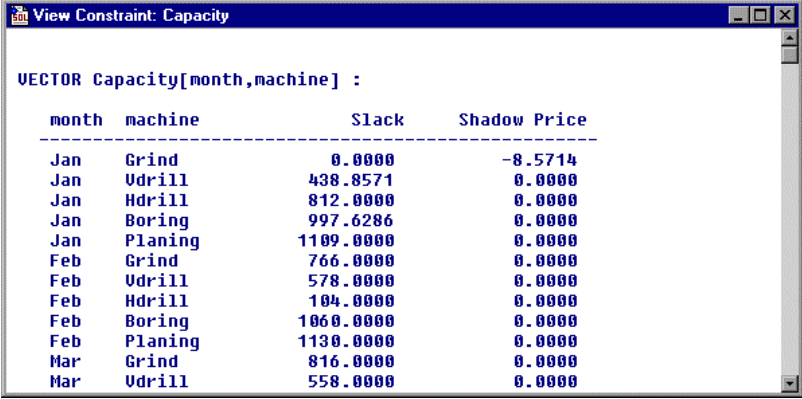
To obtain solution values, such as activity and reduced cost, for a variable, select the variable name in the tree and press the *View* button. This will open a view window showing the solution values for the selected variable. If the *Compute Ranges* option in the *General Solver Options* dialog box is *On* the sensitivity analysis ranges for the objective function coefficients are also shown. If the model has only been parsed, which means there are no solution values available, only the expanded name for each variable is shown.

The MACROS Branch

All defined macros for the current model are listed in the *MACROS* branch. If there are multiple *MACROS* sections in the model they will be shown as separate branches in the tree. To obtain more information about a macro, select the macro name in the tree and press the *View* button. This will open a view window containing the declaration and contents for the selected macro.

The CONSTRAINTS Branch

All defined constraints for the current model are listed in the *CONSTRAINTS* branch. To obtain solution values, such as slack and shadow prices, for a constraint, select the constraint name in the tree and press the *View* button. This will open a view window showing the solution values for the selected constraint.



VECTOR Capacity[month,machine] :

month	machine	Slack	Shadow Price
Jan	Grind	0.0000	-8.5714
Jan	Udrill	438.8571	0.0000
Jan	Hdrill	812.0000	0.0000
Jan	Boring	997.6286	0.0000
Jan	Planing	1109.0000	0.0000
Feb	Grind	766.0000	0.0000
Feb	Udrill	578.0000	0.0000
Feb	Hdrill	104.0000	0.0000
Feb	Boring	1060.0000	0.0000
Feb	Planing	1130.0000	0.0000
Mar	Grind	816.0000	0.0000
Mar	Udrill	558.0000	0.0000

Figure 3.9: View Constraint Solution Values

If the *Compute Ranges* option in the *General Solver Options* dialog box is *On* the sensitivity analysis ranges for the right-hand side are also shown. If the model has only been parsed, which means there is no solution available, only the expanded name for each constraint is shown.

The Message Window

While **MPL** is running it can send various progress information to a message window. Before solving or parsing a model you can open the message window by choosing *Message Window* from the *View* menu. If you want you can leave this window open while you are working in **MPL** as it will be updated automatically when you run your model.

What information is sent to the window will depend on which options are selected in the *Message Window* group of the *MPL Environment Options* dialog box.

- **Status window messages:** Sends all the status messages that are displayed in the topmost area of the *Status Window* to the *Message Window*.
- **MPL input lines:** Sends all input lines from the **MPL** model file to the *Message Window*.
- **Performance statistics:** Sends performance statistics on parsing in the **MPL** model file to the *Message Window*.
- **Memory Usage Statistics:** Sends statistics on the **MPL** parser memory usage to the *Message Window*.

- **Warning messages:** Sends all warning and error messages from **MPL** to the *Message Window*.
- **Database Connection:** Sends all messages concerning connection to databases to the *Message Window*.
- **SQL statements:** Sends all SQL statements that are issued when importing and exporting data through the database connection to the *Message Window*.
- **Solver Iteration Log:** Sends all iteration log information from the solver to the *Message Window*.

Please refer to the section on the *MPL Environment Options* dialog box in *Chapter 4.9 The Options Menu* for more information.

Using Projects to Manage Models

If you are using **MPL** to work on multiple models, that use different files and option settings, it may be beneficial to use *Projects* to manage the models. Projects are used to store information about open model files and windows, the default working directory, and all the current option settings. You can either place the project file in the same directory as the model or in a separate directory where you store all your project files.

The *Project* menu is used to open, save or close different project files. To create a new project choose the *New Project* command from the *Project* menu. This will open a dialog box where you can enter the filename for the project, the working directory, and the main **MPL** filename.

To open an existing project file choose the *Open Project* command. This will open the *Open Project* dialog box where you can open the project file. Before the selected project is opened the previous project is closed, along with all windows currently open. When a project is opened, **MPL** will re-open all the windows at the same place and size as before. Also all the option settings are set to the same values that they were the last time the project was used. **MPL** will also add the name of the project to its title bar for the main window so you can easily see which project you have open.

To close a current project choose the *Close Project* command. This will close the project along with all currently open windows, saving their place and size to the project file. All options settings will also be saved to the project file.

If you have made any changes to a project and want to save it without closing the project choose the *Save Project* command. This will save all the position and size of all currently open windows along with the current options settings. If you want to create a separate project file under a new name with all the same settings as the project currently open choose *Save As Project* command.

If you want to change properties for the current project choose the *Properties* command from the *Project* menu. This will open a dialog box where you can change properties for the project, such as the working directory or the main model filename.

If you work in **MPL** without using projects, **MPL** will maintain a default project file *mpwin.mpj* located in the *mplwin* directory to store the position and size of open windows and options settings.

Using the MPL Help System

MPL offers the user an extensive *help system* containing useful information on how to use the *MPL Modeling System*. To access the help system choose the *Topics* command from the *Help* menu. You can also enter the help system by pressing the *F1* key while running **MPL**. The help system dialog box contains three *tabs*; the *Contents* tab, the *Index* tab, and the *Find* tab, giving you different ways of accessing the help.

The Contents tab

The *Contents* tab shows all the available topics in the help system in a hierarchical tree structure. Each item shown as a book in the tree corresponds to a category or a section in the help while the items containing the actual help topics are represented by a page with a question mark on it.

You can double-click on any of the books to see all the topics available in that category or section. The **MPL** help system contains three main categories:

- The *MPL Model Development Environment* includes detailed information on how to use the **MPL** modeling environment, the available menu commands, the toolbar, and a description of all the options for each of the dialog boxes.
- The *MPL Modeling Language* contains a complete reference to the **MPL** language organized in the same way as the printed version of the manual.
- The *MPL Tutorial* contains multiple sessions with a series of models, gradually increasing in difficulty; in order to explain how to formulate linear programming models. This tutorial is specifically designed for teaching optimization modeling the way it is being applied in the corporate world.

You can continue opening books and browsing through the contents of the help until you locate a topic that looks useful.

You can print a topic by pressing the *Print* button at the bottom. If you want to print all the topics in a book, select it then press the *Print* button. Each topic will be printed on a separate page.

The Index tab

The *Index* tab enables the user to access a list of all the keywords in the help file or do a search for a specific keyword. The keyword list resembles a book index, with secondary entries indented beneath the primary entries.


As you type in a keyword, the list is automatically scrolled to match the characters typed against the primary entries in the list. If a word is not on the list, the user is taken to the word closest to where the word ought to be. Of course, you can still scroll through the keyword list and choose a word.

The Find tab

The *Find* tab enables the user to find a help topic by searching for a specific words or phrases in the text. The Find database is built the first time you select the *Find* tab through the *Find Setup Wizard*. The depth of the full-text search generated by the wizard is determined in the first wizard

window. The recommended choice, *Minimize Database Size*, produces the smallest word list, while *Maximize Search Capabilities* gives the largest database and therefore the best search capabilities. You can further customize the search by pressing the *Options* button while in the Find tab.

Context Sensitive Help

One of the nicer features of the help system in Windows is the addition of context sensitive help for dialog boxes. To display the help select the question mark button  in the upper right corner of the dialog box and then click on the item in the dialog box you want help for. A small window will pop up with a short explanation of the item you selected.

This allows you quickly get information about the item you are interested in. You can also display the context sensitive help by pressing the right mouse button on the item in the dialog box and select *What's This?* from the popup menu.

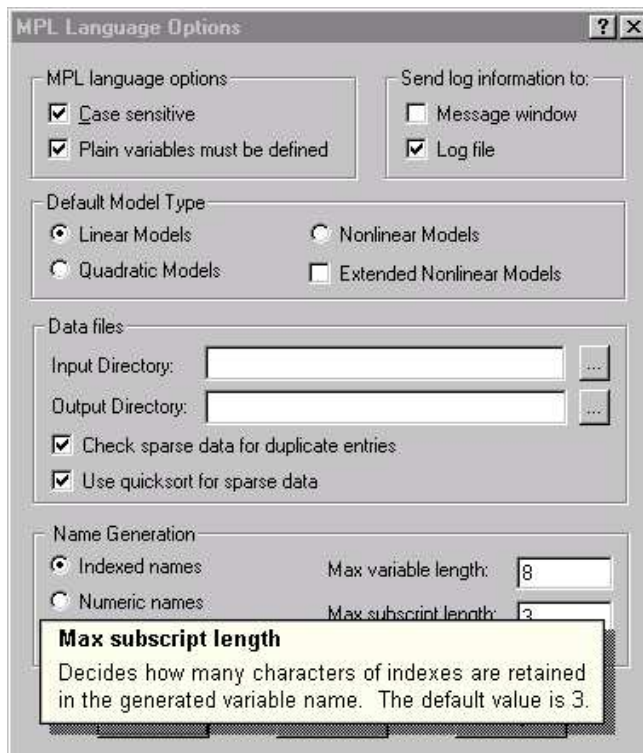


Figure 3.10: Context Sensitive Help for Dialog Box

CHAPTER 4

DESCRIPTION OF MENUS

Using the menus in **MPL** is the same as in any other Windows program; just point to the menu you wish to use and press the left mouse button to pull it down. From there you can choose the menu item you want. You will notice that some of the commands in the pull-down menu are grayed or inactive. Which menus are active depends on what window is currently at the top and the state of the program.

Also, notice that some of the commands in the menus have an ellipsis ‘...’ after them, indicating that you have to supply more information in a dialog box before **MPL** can carry out the command.

Some of the menu items are followed by a Ctrl-key entry. These Ctrl-key combinations are there to provide the user with a short-cut. Instead of pulling down menus and choosing the commands, you can just press Ctrl and the appropriate key at the same time.

This chapter provides a reference to all the menu items available in **MPL**.

4.1 The Main Menu

At the top of the main screen in **MPL** the main menu resides, from which you select the actions you wish to perform.

File	- Open, save, and print files
Edit	- Undo, cut, copy, and paste commands
Search	- Search, replace, goto line commands
Run	- Run solver, check syntax, and generate input
Project	- Open, save, and close projects
View	- View the solution on screen
Graph	- Display graph of the matrix and the objective
Options	- Change default option for MPL
Window	- Handling of multiple windows
Help	- On-line help for MPL

In the following pages, each menu item will now be described in full detail.

4.2 The File Menu

The *File* menu is used to open, save, and print model files. To access the *File* menu, simply point at the word *File* in the main menu and hold down the mouse button. The menu appears as shown here in Figure 4.1.



Figure 4.1: The File Menu

New	- Create a new MPL model file
Open...	- Open an existing model file
Close	- Close the current editor window
Save	- Save the current model file to disk
Save As...	- Save model file under a new name
Save Selection...	- Save selected text to disk
Insert File...	- Insert file from disk into current model
Print	- Print the current model file
Exit	- Quit MPL

Create a New Model File

If you need to create a new model file, choose the *New* command from the *File* menu and **MPL** will open an empty editor window with the name *Untitled*. If another window is already open, **MPL** will place the new window on top of the existing one. When you are finished editing and want to save it to a file **MPL** will ask for the filename.

Open an Existing Model File

To open an existing model file, pull down the *File* menu and choose the *Open* command. You can also use the shortcut *Ctrl-O* or press the *File Open* button in the Toolbar. The standard *Open File* dialog box like the one shown below in Figure 4.2 will then be displayed.

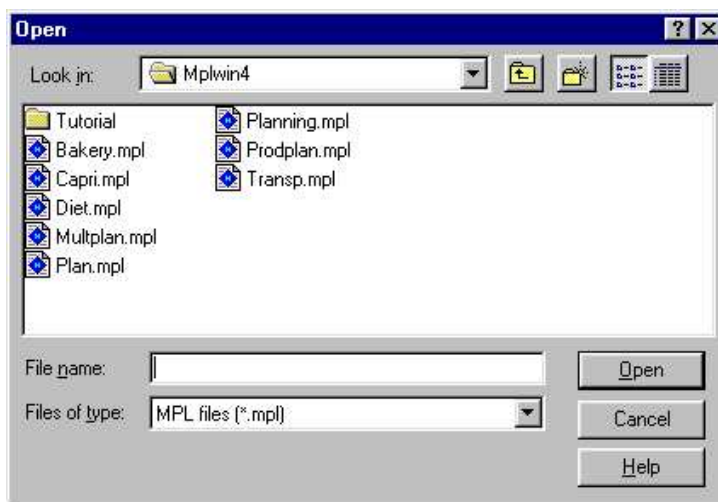


Figure 4.2: The Open File Dialog Box

In the middle of the dialog box is a list of the files that are in the current folder. To open a file, select it from the list of files and press the *Open* button. Alternatively, you can enter the filename in the *File name* input box below the list. You can also open the file directly by double-clicking on it in the list of files. If you type in a file name that does not exist, **MPL** will automatically create a new empty file with that name.

The files that are listed in the list of files are determined by the entry in the *Files of type* input box located at the bottom. From there you can select to show **MPL** model files '*.mpl', MPS files '*.mps', data files '*.dat', project files '*.mpj', or all files '*.*'

The name of the current folder is shown in the *Look In* input box at the top of the dialog. If the file you want to open is in a different folder, you can press the down arrow at the right of the *Look In* input box to navigate through the directory tree. Select the folder name you want to go to and the list of files below will reflect the contents of the new folder.

If you do not want to open a file press the *Cancel* button to close the dialog box.

The maximum file size for the editor in **MPL** is about 40-50K. If the file you are trying to open is too big, **MPL** will truncate the file and change the editor window to a view only mode. For large models, you should consider using include files which are described in the *Include Files* section in *Chapter 5.2: Basic Input Elements*.

Close Files

To close the file in the current window, choose *Close* from the *File* menu. If you have changed or edited the model since it was last saved **MPL** will display a question dialog box that will ask whether you want to save the file before you close it.



Figure 4.3: Save File Before Closing Question

If you want to save the file click on the *Yes* button. If you do not want to save the file click on the *No* button and any changes you have made to the file will be lost.

Save the Model File

MPL has two commands on the *File* menu *Save* and *Save As...* that allow you to save your model files. You can also use the shortcuts *Ctrl-S* and *Ctrl-A* respectively or press the *File Save* button in the Toolbar.

You use the *Save* command when you want to save the file using the current name. When you want to save the file under a different name use the *Save As...* command. The standard *Save As* dialog box like the one shown below will then be displayed.

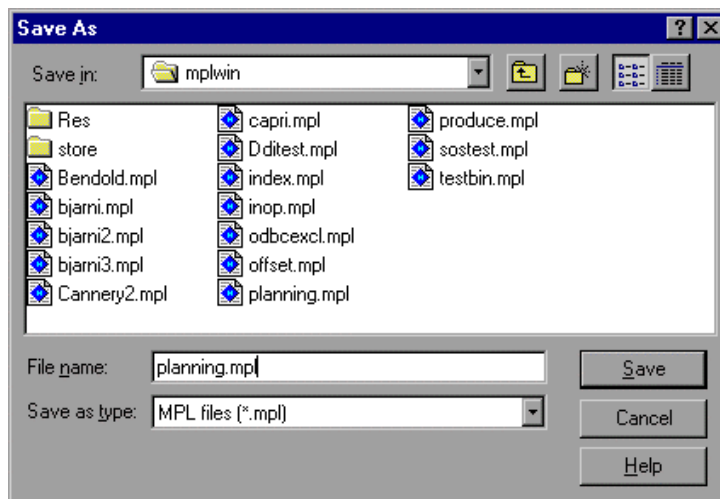


Figure 4.4: The Save As Dialog Box

In the middle of the dialog box is a list of the files that are in the current folder. Enter the new filename you want to save as in the *File Name* input box. You can also save the file by selecting one of the existing filenames in the list of files and then press the *Save* button.

The files that are listed in the list of files are determined by the entry in the *Save as type* input box located at the bottom. From there you can select to show **MPL** model files **.mpl*, MPS files **.mps*, data files **.dat*, project files **.mpj*, or all files **.**.

The name of the current folder is shown in the *Save In* input box at the top of the dialog. If you want to save the file in a different folder, you can press the down arrow at the right of the *Save In* input box to navigate through the directory tree. Select the folder name you want to save in, then enter the new filename in the *File Name* input box and press the *Save* button.

If you do not want to save the file press the *Cancel* button to close the dialog box.

Save Selected Text to a File

MPL allows you to select text in the editor and save to a separate file by choosing the *Save Selection* command in the *File* menu. This will bring up the *Save Selection* dialog box shown below.

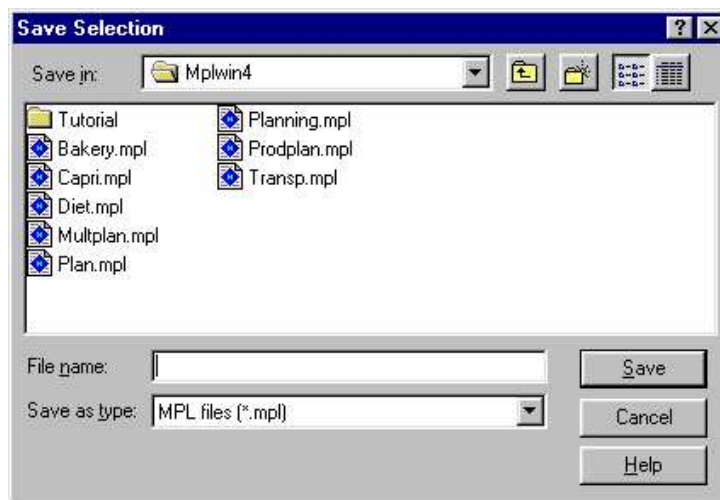


Figure 4.5: The Save Selection Dialog Box

This dialog box is very similar to the *Save As* dialog box explained on the previous page. In the middle of the dialog box is a list of the files that are in the current directory. Enter the filename you want to save the selection as in the *File Name* input box and then press the *Save* button.

The files that are listed in the list of files are determined by the entry in the *Save as type* input box located at the bottom. From there you can select to show **MPL** model files **.mpl*, MPS files **.mps*, data files **.dat*, project files **.mpj*, or all files **.**.

The current folder is shown in the *Save in* input box on the top of the dialog. If you want to save the selection in a different folder, you can press the down arrow at the right of the *Save in* input box to navigate through the directory tree. Select the folder name you want to save in, then enter the new filename in the *File Name* input box and press the *Save* button.

If you do not want to save the selection press the *Cancel* button to close the dialog box.

Insert File Into the Editor

MPL allows you to insert a file into the current editor file by choosing the *Insert File* command from the *File* menu. This will bring up the *Insert File* dialog box.

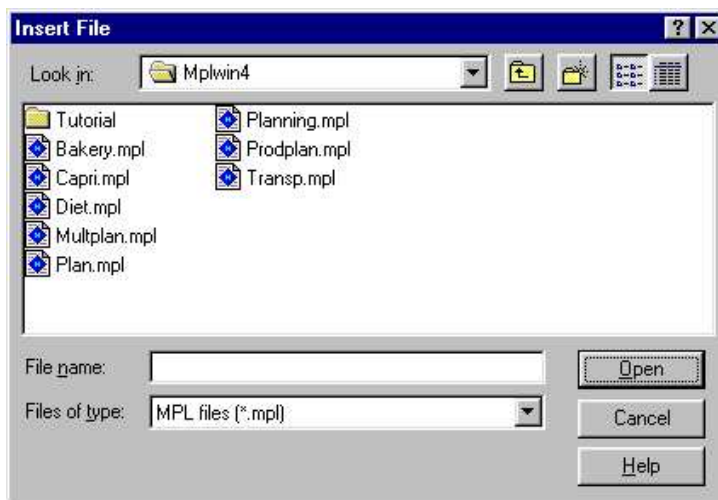


Figure 4.6: The Insert File Dialog Box

This dialog box is very similar to the *Open* dialog box explained in a previous section. In the middle of the dialog box is a list of the files that are in the current folder. Enter the name of the file you want to insert in the *File Name* input box and press the *Open* button.

The files that are listed in the list of files are determined by the entry in the *Files of type* input box located at the bottom. From there you can select to show **MPL** model files **.mpl*, MPS files **.mps*, data files **.dat*, project files **.mpj*, or all files **.**.

The current folder is shown in the *Look in* section at the top of the dialog. If you want to save the selection in a different folder, you can press the down arrow at the right of the *Look in* input box to navigate through the directory tree. Select the folder name you want to save in and the list of files below will reflect the contents of the new folder.

After you have selected the file you want to insert, press the *Open* button to insert it. You can also insert the file by double clicking on the file name in the list of files.

If you do not want to insert the file press the *Cancel* button to close the dialog box.

Print the Model File

To print the model file in the current editor window, choose *Print* from the *File* menu. You can also use the shortcut *Ctrl-P* or press the *File Print* button in the Toolbar. The standard *Print* dialog box like the one shown below in Figure 4.7 will then be displayed.

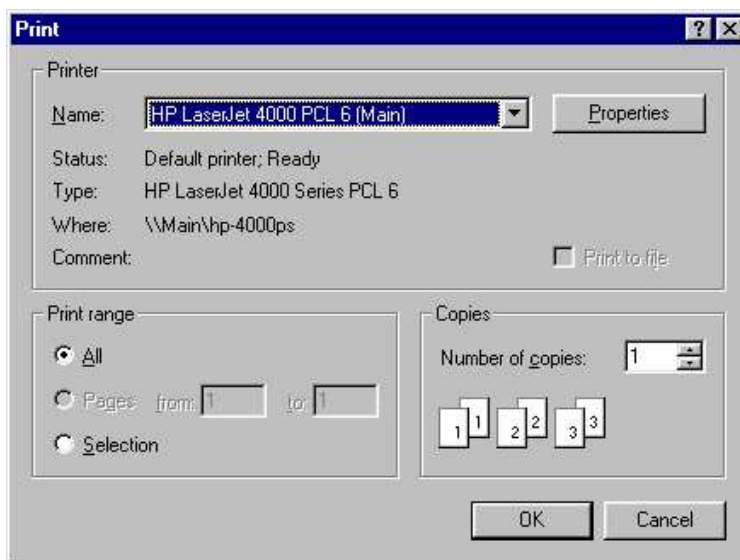


Figure 4.7: The Print File Dialog Box

Select whether you want to print all the text in the editor, the selected text only, or pages only from the given from/to range. The default printer, that will be used, is shown in the top section of the dialog box. If you want to change the default printer, press the down arrow at the right of the *Name* input box to navigate through the printer selections. Select the printer name you want to go to and the *Name* input box will reflect your new printer.

Exit the MPL Program

To exit the **MPL** program, choose *Exit* from the *File* menu. If you have changed or edited any of the model files that are open, **MPL** will display a question dialog box for each of them that will ask you whether you want to save the file before it is closed.



Figure 4.8: Save File Before Quitting MPL Question

If you want to save the file, click on the *Yes* button. If you do not want to save the file click on the *No* button. Any changes you have made to the file will be lost. If you do not want to exit **MPL** you can cancel the operation by pressing the *Cancel* button.

4.3 The Edit Menu

The *Edit* menu is used to cut, copy, and paste with the clipboard and to undo changes you have made to your model file.



Figure 4.9: The Edit Menu

Undo	- Undo the last change made to model file
Cut	- Move the selected text to the clipboard
Copy	- Copy the selected text to the clipboard
Paste	- Insert text from the clipboard
Delete	- Delete the selected text in the model file
Select All	- Select all the text in the model file

Undo Changes

If you make a change to your model file that you want to take back, choose *Undo* from the *File* menu. You can also use the shortcut *Ctrl-Z* to undo your changes.

Clipboard Operations

The clipboard in Windows allows you to copy text either within the editor or between different editor files. You can also use the clipboard to copy text between different applications. To copy text to the clipboard, you first select the text, then you copy it to the clipboard using the *Copy* command. If you want to move the text instead of copying it, you can use the *Cut* command. After you have copied the text to the clipboard, you point the cursor where you want to place the text and use the *Paste* command to place it.

MPL supports all the text selection methods that are standard in Windows. To select text, simply press the left mouse button where you want the selection to start and drag the cursor to the end of the text you want to select. If you want to extend the current selection, hold down the *Shift* key as you press down the left mouse button. To select all the text in the file you, choose *Select All* from the *Edit* menu.

After you have selected the text, you can either cut or copy the text to the clipboard. If you want to cut the text from the editor, choose *Cut* from the *Edit* menu. You can also use the shortcut *Ctrl-X* or press the *Edit Cut* button in the toolbar to cut it. This will move the text from the editor and place it in the clipboard. If you want to copy the text, choose *Copy* from the *Edit* menu. You can also use the shortcut *Ctrl-C* or press the *Edit Copy* button in the toolbar to copy it. This will copy the text from the editor and place it in the clipboard.

To place text from the clipboard into the editor, first make sure the cursor is where you want the text to be placed. Then choose the *Paste* command from the *Edit* menu to place the text. You can also use the shortcut *Ctrl-V* or press the *Edit Paste* button in the toolbar to place it.

If you only want to remove the selected text without placing it in the clipboard, you can choose the *Delete* command from the *Edit* menu. You can also use the shortcut *Ctrl-Del* to delete it.

4.4 The Search Menu

The *Search* menu is used to find and replace text, and go to a specific line in the model file.



Figure 4.10: The Search Menu

- | | |
|-------------------|--|
| Find... | - Search for text in the model file |
| Replace... | - Replace text in the model file |
| Next | - Search for the next occurrence of text |
| Goto Line | - Goto specific line in the model file |

Find Text in the Model

To search for text in the model file, choose *Find* from the *Search* menu. You can also use the shortcut *Ctrl-F* or press the *Search Find* button in the Toolbar. The *Find* dialog box like the one shown below in Figure 4.11 will then be displayed.



Figure 4.11: The Find Dialog Box

Enter the text you want to search for in the *Search for* input box. **MPL** offers a number of options that control how it searches for text. These options are described below:

Options

Case sensitive: If *On* search only for text that has exactly the same case as the text given in the above Search for input box. If *Off*, the text is found regardless of the case.

Whole words only: If *On* search only for whole words in the text. If *Off*, text that is part of another word will also be found.

Direction

Forward: Search forward from the current position.

Backward: Search backward from the current position.

Origin

From cursor: Start the search from the current position.

Entire scope: Search for the entire scope of the file.

After you have entered the search text and the options, press the *OK* button to start the search. To repeat the search select *Next* from the *Search* menu, or press the *Search Next* button in the Toolbar.

Find and Replace Text

To replace text in the model file, choose *Replace* from the *Search* menu. You can also use the shortcut *Ctrl-R* or press the *Search Replace* button in the Toolbar. The *Find and Replace* dialog box like the one shown below in Figure 4.12 will then be displayed.

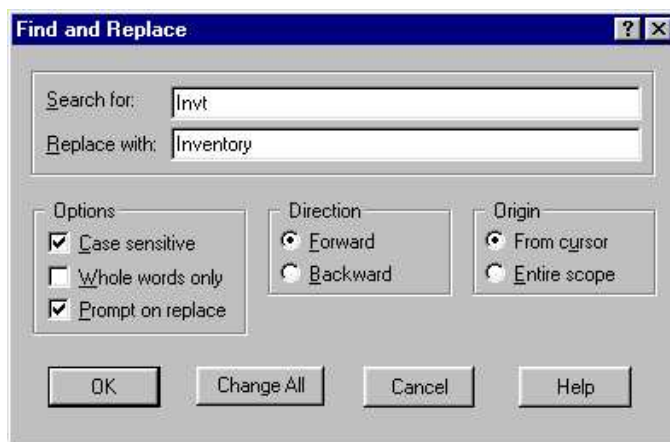


Figure 4.12: Find and Replace Dialog Box

Enter the text you want to search for in the *Search for* input box and the text you want to replace it with in the *Replace with* input box. **MPL** offers a number of options that control how it searches and replaces text. These options are described below:

Options

Case sensitive: If *On*, search only for text that has exactly the same case as the text given in the above *Search for* input box. If *Off*, the text is found regardless of the case.

Whole words only: If *On*, search only for whole words in the text. If *Off*, text that is part of other word will also be found.

Prompt on replace: If *On*, you will be prompted for each replace. If *Off*, the replacement will be automatic.

Direction

Forward: Search forward from the current position.

Backward: Search backward from the current position.

Origin

From cursor: Start the search from the current position.

Entire scope: Search for the entire scope of the file.

After you have entered the search and replace text and the options press the *OK* button to start the replace. To repeat the replace select *Next* from the *Search* menu, or press the *Search Next* button in the Toolbar. If you want to replace all the occurrences of the text in one operation press the *Replace All* button.

Goto Line in the Model

To go to a specific line in the model file, choose *Goto Line* from the *Search* menu. You can also use the shortcut *Ctrl-G* or press the *Search Goto Line* button in the Toolbar. The *Goto Line* dialog box like the one shown below in Figure 4.13 will then be displayed.

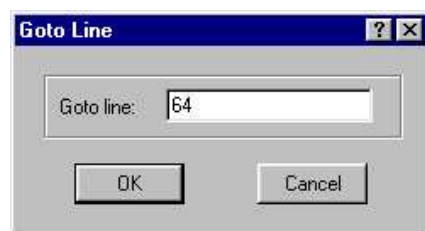


Figure 4.13: The Goto Line Dialog Box

Enter the line you want to go to in the model file and press the *OK* button. If you want to cancel the operation, press the *Cancel* button

4.5 The Project Menu

The *Project* menu is used to open, save, and close projects. To access the *Project* menu, simply point at the word *Project* in the main menu and hold down the mouse button. The menu appears as shown here in Figure 4.14.



Figure 4.14: The Project Menu

New Project	- Create a new project file
Open Project...	- Open an existing project file
Close Project	- Close the current project
Save Project	- Save the current project to disk
Save as Project	- Save project under a new name
Properties	- Changing properties for the current project

Create a New Project File

If you need to create a new project, choose the *New Project* command from the *Project* menu and **MPL** will open the *New Project* dialog box. You can indicate the project filename, the working folder and the main **MPL** filename in the boxes provided.

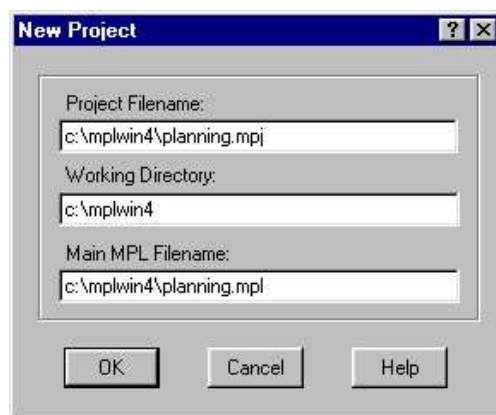


Figure 4.15: The New Project Dialog Box

If you want to create and save the project click on the *OK* button. If you do not want to save the project then click on the *Cancel* button.

Open an Existing Project File

To open an existing project file, pull down the *Project* menu and choose the *Open Project* command. You can also press the *Project Open* button in the Toolbar. The standard *Open Project* dialog box, like the one shown on the next page in Figure 4.16, will then be displayed.

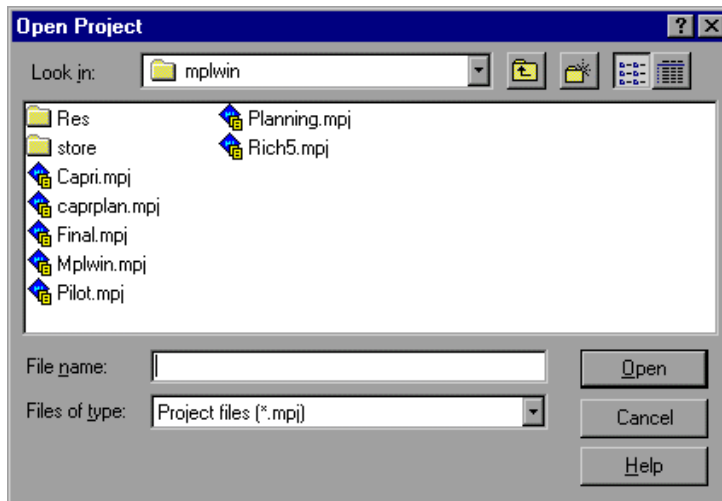


Figure 4.16: The Open Project File Dialog Box

Enter the name of the project file you want to open in the *File Name* input box and click on *Open*. Alternatively, you can also use the list box in the middle of the dialog to select a file in the current folder and open it by double clicking on the filename with the mouse.

The files that are listed in the list of files are determined by the entry in the *Files of type* input box located at the bottom of the dialog. From there you can select to show **MPL** project files '*.mpi', or all files '*.*'

The name of the current folder is shown in the *Look In* input box at the top of the dialog. If the project file you want to open is in a different folder, you can press the down arrow at the right of the *Look In* input box to navigate through the directory tree. Select the folder name you want to go to and the list of project files below will reflect the contents of the new folder.

If you do not want to open a project file press the *Cancel* button to close the dialog box.

Close Project Files

To close the project file in the current window, choose *Close Project* from the *Project* menu. If you have changed or edited the project file since it was last saved **MPL** will display a question dialog box that will ask whether you want to save the project file before you close it.

If you want to save the file click on the *OK* button. If you do not want to save the project file click on the *Cancel* button and any changes you have made to the project file will be lost.

Save the Project File

MPL has two commands on the *Project* menu *Save Project* and *Save As Project* that allow you to save your project files.

You use the *Save Project* command when you want to save the project file using the current name. You can also press the *Project Save* button in the Toolbar. When you want to save the project file under a different name use the *Save As Project* command. The standard *Save As Project* dialog box, like the one shown below, will then be displayed.

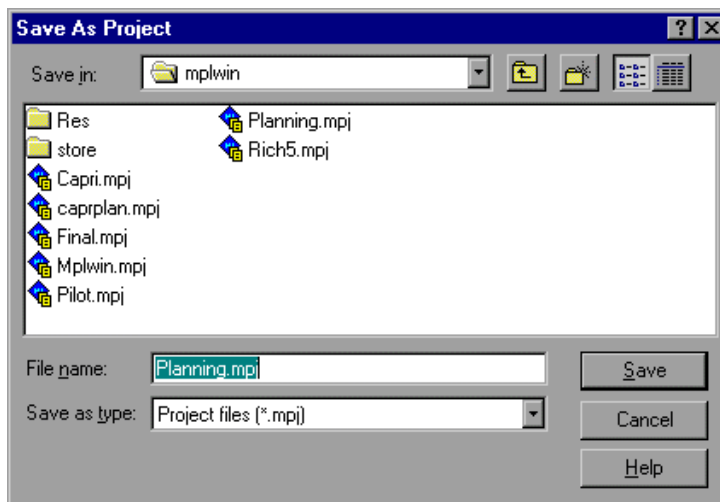


Figure 4.17: The Save As Project Dialog Box

In the middle of the dialog box is a list of the project files that are in the current folder. Enter the new project filename you want to save as in the *File Name* input box. You can also save the file by selecting one of the existing filenames in the list of project files and then press the *Save* button.

The files that are listed in the list of project files are determined by the entry in the *Save as type* input box located at the bottom of the dialog. From there you can select to show **MPL** project files '*.mpj', or all files '*.*'.

The name of the current folder is shown in the *Save In* input box at the top of the dialog. If you want to save the project file in a different folder, you can press the down arrow at the right of the *Save In* input box to navigate through the directory tree. Select the folder name you want to save in, then enter the new project filename in the *File Name* input box and press the *Save* button.

If you do not want to save the project file press the *Cancel* button to close the dialog box.

Change Properties for a Project

If you need to change properties for the current project, choose the *Properties* command from the *Project* menu and **MPL** will open the *Project Properties* dialog box. You can change the working directory and the main **MPL** filename for the project in the boxes provided.

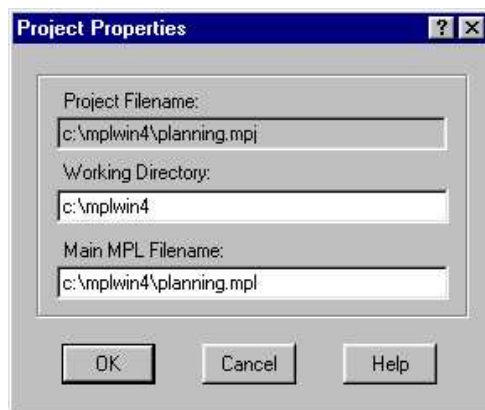


Figure 4.18: The Project Properties Dialog Box

If you want to save the changes click on the *OK* button. If you do not want to save the changes click on the *Cancel* button.

4.6 The Run Menu

The *Run* menu is used when you want to solve the model, check the syntax, and generate input files.

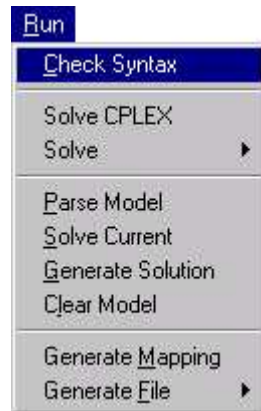


Figure 4.19: The Run Menu

Check Syntax	- Check syntax of the current MPL model file
Solve <solvername>	- Solve the model using the specified solver
Parse Model	- Parse the model file into memory
Solve Current	- Solve the model again without parsing
Generate Solution	- Generate solution file for last solver run
Clear Model	- Clear the current model from memory
Generate Mapping	- Generate mapping file for the solution
Generate File	- Generate input file for external solvers

Check Syntax of the Model

After you have entered your formulation in the model editor, you can check the model for syntax errors by choosing *Check Syntax* from the *Run* menu. You can also press the *Run Check Syntax* button in the Toolbar. If **MPL** finds a mistake in the formulation it will report it in the *Error Message* window.



Figure 4.20: The Error Message Window

The above message tells you that the index *month* in the variable vector *Inventory*, was not defined in the underlying constraint *InvBal*. The mistake was located in line 44 of the model file.

Pressing the *OK* button or the *Return* key returns you to the model editor. The cursor will automatically be positioned at the location of the error in the model file, with the offending word or character highlighted.

If you need more help on the error message, press the *Help* button. This will give you further explanation of the error, including examples. For a list of all the error messages in **MPL**, please refer to *Appendix B: Error Messages*.

Solve the Model

You can solve the model by choosing *Solve <solvername>* from the *Run* menu. The solver that is in the solve command is the current *default* solver for **MPL**. You can also run the default solver by pressing the *Run Solve* button in the Toolbar. The solve command performs several steps, including parse the model into memory, check the syntax, create the matrix for the solver, call the solver, and then create the solution file.

While optimizing, the Status Window appears, providing information about the solution progress. For details about the contents of the Status Window, refer to *Chapter 4.3: Using the MPL Environment*.

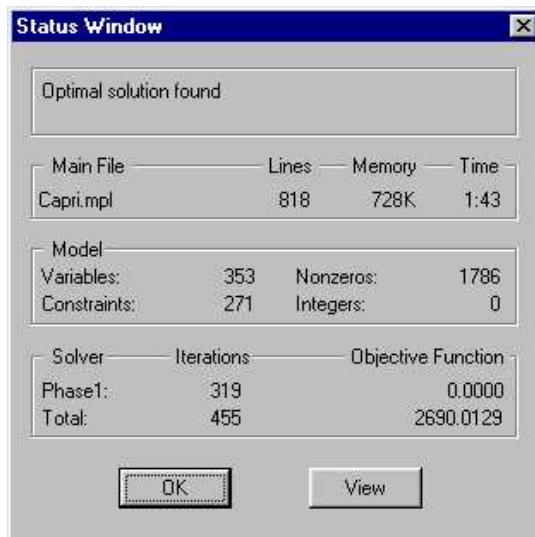


Figure 4.21 : The Status Window Dialog Box

MPL supports multiple solvers that are listed in the *Solve* submenu of the *Run* menu. To use a different solver go to the submenu and choose the solver you want to run. **MPL** will automatically update the default solver in the menu to the last one used.

The solvers **MPL** supports can be categorized into two groups, Windows DLL solvers and legacy DOS solvers. When **MPL** uses Windows DLL solvers the matrix is sent directly through memory to the solver. There are now many industry-standard solvers available as DLL's including CPLEX, XPRESS, OSL, FortMP, XA, and CONOPT, all of which are supported by **MPL**.

When **MPL** uses DOS legacy solvers, it writes out the matrix to an input file, normally an MPS file, and then starts the solver in a DOS window. When the solver has finished optimizing, **MPL** will read back the output file from the solver and return back to Windows. This method is of course not as fast as transferring the matrix through memory, but it works with almost any solver that can run in a DOS window.

If you need to change some of the setup options for a solver, choose *Solver Menu* from the *Options* menu. From the *Solver Menu Setup* dialog choose the solver you need to change setup options for in the list box and press the *Edit* button. This will display the *Solver Setup Options* dialog box. Please refer to *Chapter 4.9: The Options Menu* for how to change the setup options.

Some solvers are also provided with option dialog boxes that allow you to change various algorithmic options. Check the *Solver Parameters* dialog box in the *Options* menu to see if an option dialog box is available for your solver. If it is, please refer to the solvers documentation for more information on these options.

Parse the Model Into Memory

If you want to parse or read the model formulation into memory, without solving it, choose *Parse Model* from the *Run* menu. This will let **MPL** read through the model file, check the syntax, and store it in memory. This is especially useful when you want to analyze the model or display a graph of the matrix.

Solve the Model Currently in Memory

If you have parsed the model into memory or you have solved your model once and want to solve it again, perhaps after having changed some options, you can do so by choosing *Solve Current* from the *Run* menu. This will solve the model currently in memory, without parsing it again, which is unnecessary if the model has not been changed.

Generate Solution File

If you have changed any of the solution file options since the last solver run, you can generate the solution file again without having to solve again by choosing *Generate Solution* from the *Run* menu.

Clear the Current Model From Memory

To clear the current model from memory choose *Clear Model* from the *Run* menu. This will clear all memory used by **MPL** to store the model, solver information, the solution and other stored items such as iteration information.

Generate Mapping

When using **MPL** with other programs it is sometimes advantageous to know how variable names in **MPL** are mapped to the matrix that is sent to the solver. The mapping file contains, for each variable and constraint, the abbreviated name that is sent to the solver, the original **MPL** name, and all of the subscripts for vector entries. This enables the user to map the abbreviated name to the **MPL** name and vice versa. This file can also be automatically generated while solving by selecting *Mapping File* in the *General Solver Options* dialog box.

Generate Input File

Although, **MPL** can run most solvers directly from the menus, you might, in some circumstances, want to run the solver separately. **MPL** can be used to generate several different input formats which can be selected from the *Generate* submenu in the *Run* menu. While generating, the Status Window appears, providing information about the generation. For more information about the Status Window, refer to *Chapter 4.3: Using the MPL Environment*.

Available input formats include the standard MPS format, native input formats for several different solvers including CPLEX, XA, Lindo, and Turbo-Simplex. **MPL** can also generate an input file that is compatible with **MPL**. This can be particularly useful when you have a standard MPS file and want to create an **MPL** compatible input file.

4.7 The View Menu

The *View* menu is used after you have solved your model to view generated files and various parts of the solution in a window. You can also get problem statistics and defined items in the model.



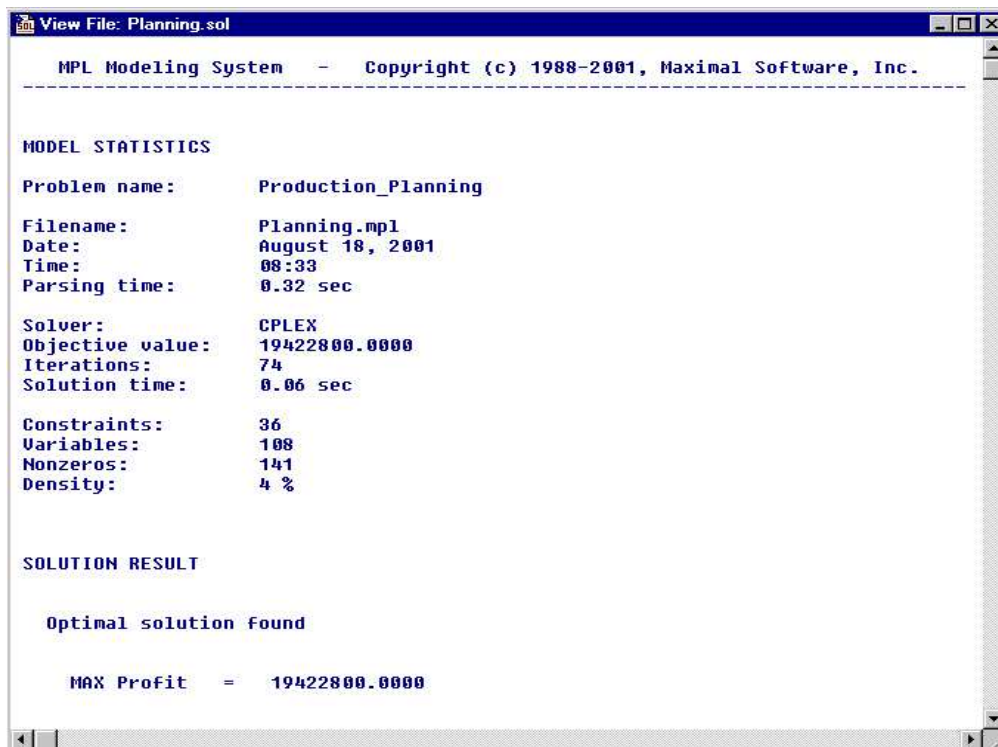
Figure 4.22: The View Menu

Files <solution file>	- View the MPL solution file generated
Files <output file>	- View the output file from the solver
Files <input file>	- View the input file for the solver
Files <log file>	- View the log file for the solver run
Files Other Files	- View any other file on the disk
Values/Reduced Cost	- View variable values and reduced costs
Slack/Shadow Prices	- View constraint slacks and shadow prices
Range Objective	- View ranges for the objective function
Range RHS	- View ranges for the right-hand-side
Model Statistics	- View the statistics of the current model
Model Definitions	- Open the model definition window
Message Window	- Open the message window

The view window stores the file in memory allowing you to quickly browse through the solution on the screen using the mouse. You can have multiple view windows open at the same time. The view window can handle very large solution files, up to several megabytes, depending on how much memory you have available on your machine. If the solution file becomes too large, you can change the contents of the file. Please refer to the *Change Solution File Options* section in *Chapter 4.9: The Options Menu* for more information.

View the Solution Files

MPL generates a solution file after solving the model. You can view this file by choosing *Files <solution filename>* from the *View* menu. The filename listed will be the name of the solution file, usually with a *.sol* extension, for example as in the model shown below *planning.sol*. You can also view the solution file by pressing the *View Solution File* button in the Toolbar.



```
View File: Planning.sol
MPL Modeling System - Copyright (c) 1988-2001, Maximal Software, Inc.
-----
MODEL STATISTICS
Problem name:      Production_Planning
Filename:          Planning.mpl
Date:              August 18, 2001
Time:              08:33
Parsing time:      0.32 sec

Solver:            CPLEX
Objective value:   19422800.0000
Iterations:        74
Solution time:     0.06 sec

Constraints:       36
Variables:         108
Nonzeros:          141
Density:           4 %

SOLUTION RESULT

Optimal solution found

MAX Profit = 19422800.0000
```

Figure 4.23: Viewing the Solution File in a Window

When you solve the model with a legacy DOS solver both a solver input file (usually MPS) and an output file will be generated. Both of these files will be listed also in the *View Files* submenu so you can view them. If something goes wrong during the optimization process these files can be very useful identifying the problem. Please refer to the documentation that came with the solver for the description of the input and output file formats.

View Other Files

To view any file other than solution files, pull down the *View* menu and choose the *Files / Other Files* command. The *View File* dialog box, like the one shown below in Figure 4.25, will then be displayed.

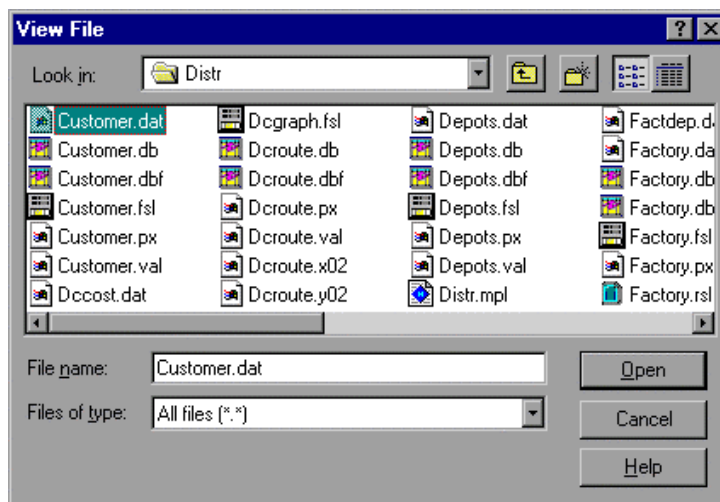


Figure 4.24: The View File Dialog Box

In the middle of the dialog box is a list of the files that are in the current folder. To open a file, select it from the list of files and press the *Open* button. You can also open the file directly by double-clicking on it in the list of files. Alternatively, you can enter the filename in the *File name* input box below the list and then press the *Open* button.

The name of the current folder is shown in the *Look In* input box at the top of the dialog. If the file you want to open is in a different folder, you can press the down arrow at the right of the *Look In* input box to navigate through the directory tree. Select the folder name you want to go to and the list of files below will reflect the contents of the new folder.

If you do not want to open a file press the *Cancel* button to close the dialog box.

View Solution Values

MPL allows you to view various solution values, in a view window without having to include them in the solution file. There are four possible tables to view:

1. For the activity values and the reduced costs for each decision variable in the optimal solution choose *Values/Reduced Cost* from the *View* menu.
2. For the slacks and the shadow prices for each constraint in the optimal solution choose *Slacks/Shadow Prices* from the *View* menu.
3. For the ranges of the objective function coefficients choose *Ranges Objective* from the *View* menu. The resulting table will contain the name of the variable, the objective function coefficient, the lower bound, and the upper bound.
4. For the ranges of the right hand side values choose *Ranges RHS* from the *View* menu. The resulting table will contain the name of the constraint, the RHS value, the lower bound, and the upper bound.

Please note that the *Compute Ranges* option in the *Solution File Options* dialog box must be set *prior* to solving the model if you want to be able to view ranges. Below, in Figure 4.26 is an example of the view window for ranges of the objective function coefficients.

The screenshot shows a window titled "View: Ranges Objective Function" with the following content:

```

RANGES OBJECTIVE

VECTOR Inventory[product,month] :

  product month      Coefficient    Lower Bound    Upper Bound
-----
  1  January         -8.8000        -1E+20         0.0000
  1  February        -8.8000        -1E+20         0.0000
  1  March           -8.8000        -1E+20         0.0000
  1  April           -8.8000        -1E+20         0.0000
  1  May             -8.8000        -1E+20         0.0000
  1  June            -8.8000        -1E+20         0.0000
  1  July            -8.8000        -1E+20         0.0000
  1  August          -8.8000        -1E+20         0.0000
  1  September       -8.8000        -1E+20         0.0000
  1  October         -8.8000        -1E+20         0.0000
  1  November        -8.8000        -1E+20         0.0000
  1  December        -8.8000        -1E+20         1E+20
  2  January         -8.8000        -1E+20        -1.9000
  2  February        -8.8000        -10.7000      -1.9000
  2  March           -8.8000        -10.7000      -1.9000
  2  April           -8.8000        -10.7000      -1.9000
  2  May             -8.8000        -10.7000      -1.9000
  2  June            -8.8000        -10.7000      -1.9000
  2  July            -8.8000        -1E+20         0.0000
  2  August          -8.8000        -1E+20         0.0000
    
```

Figure 4.25: The View Ranges Objective Window

View the Model Statistics

To see the model statistics choose *Model Statistics* from the *View* menu. This will display the *Model Statistics* dialog box shown below in Figure 4.27.

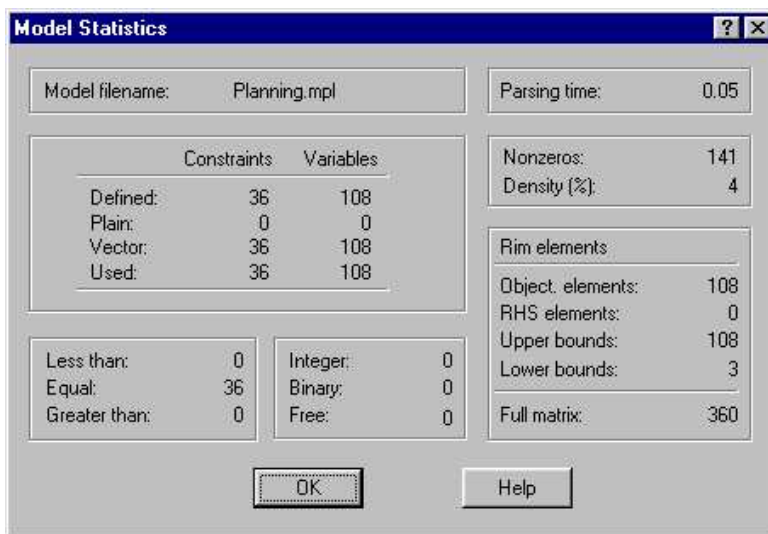


Figure 4.26: Viewing Model Statistics

The top line of the model statistics window contains the names of the model file. On the right is the time it took to parse the model.

On the left, in the middle of the window, is a table containing the number of constraints and variables. If you defined variables in the *DECISION VARIABLES* section of the model, but didn't use them all in the model, the *Defined* variable count will be higher than the *Used* variables. The *Plain* and *Vector* variable count tells you how many of the variables were plain and how many were defined as vectors.

At the bottom of the window, to the left, the constraints are divided into *Less Than*, *Equal*, and *Greater Than* constraints. In the next section, to the right, the number of variables that are *Free*, *Integer*, or *Binary* variables are listed.

To the right, in the middle of the window, are the number of nonzeros and the matrix density. Below, in the lower right-hand corner, the number of objective function coefficients, the right hand side (RHS) coefficients, and the upper and lower bounds are listed. Finally, at the bottom, it shows the total number of nonzero elements for the full matrix that were stored.

View Model Definitions Window

MPL allows you to view defined items from the model formulation in a tree window. Choose *Model Definitions* from the *View* menu, to display the *Model Definitions* window. You can also press the *View Model Definitions* button in the Toolbar. You can leave this window open while you are working in MPL as it will be updated automatically when you parse in your model.

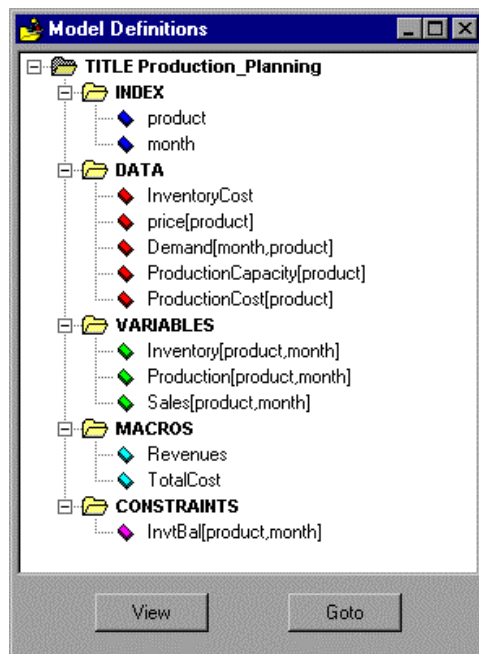


Figure 4.27: The Model Definitions Window

This window will show all the defined items in the model in a hierarchical tree structure. Each branch in the tree corresponds to a section in the model. You can expand and collapse each branch to show only the elements you are interested in.

At the bottom of the window are two buttons. The *View* button allows you to display the contents of each defined item in a separate view window. What is displayed for each item is discussed in the following pages. The *Goto* button will take you directly to the declaration of the selected item in the model file.

MPL has several options in the *MPL Environment Options* dialog box that allow you to control how the Model Definitions window behaves. You specify whether each defined item is shown with the number of elements it contains. You can also specify whether each branch is expanded or collapsed when the tree is initially created. Finally you can set whether double-clicking on a defined item will work as the *Goto* or the *View* button.

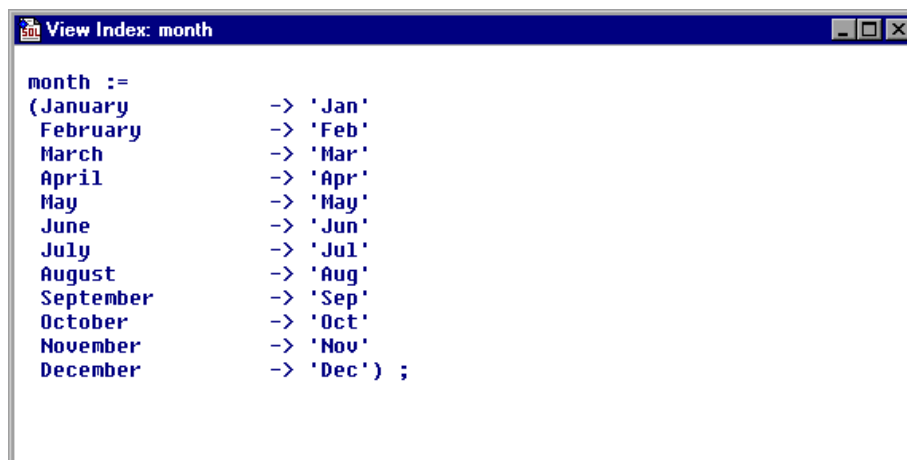
The INDEX Branch

All defined indexes for the current model are listed in the *INDEX* branch. If there are multiple *INDEX* sections in the model they will be shown as separate branches in the tree.

To obtain more information about an index entry, select the index name in the tree and press the *View* button. This will open a view window containing the declaration and contents for the selected index entry. What is shown for each index depends on how it was declared in the model.

- For *numeric indexes* the lower and upper range values are shown.
- For *named indexes* the name of each index element is shown along with the abbreviated name which is used to generate variable and constraint names.
- For *multi-dimensional indexes* all the index elements are shown along with the shorter abbreviated name similar to named indexes.

For example to view the named index *month* select the name in the *INDEX* branch and press the *View* button. This will display the view window shown here below:



```

month :=
(January      -> 'Jan'
 February     -> 'Feb'
 March        -> 'Mar'
 April        -> 'Apr'
 May          -> 'May'
 June         -> 'Jun'
 July         -> 'Jul'
 August       -> 'Aug'
 September    -> 'Sep'
 October      -> 'Oct'
 November     -> 'Nov'
 December     -> 'Dec') ;

```

Figure 4.28: View Named Index Elements

The DATA Branch

All defined data vectors and data constants for the current model are listed in the *DATA* branch. If there are multiple *DATA* sections in the model they will be shown as separate branches in the tree.

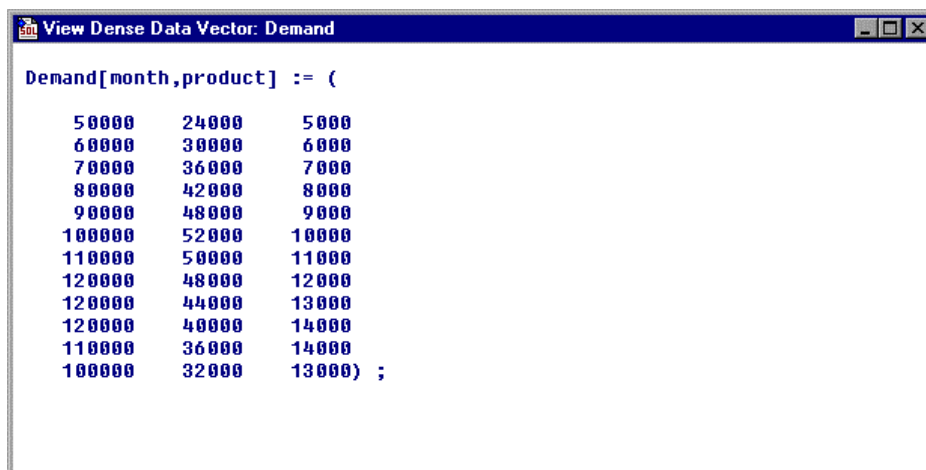
To obtain more information about a data entry, select the data name in the tree and press the *View* button. This will open a view window containing the declaration and contents for the selected data entry.

Part II Using the MPL Modeling System

What is shown for each data entry depends on how it was declared in the model.

- For *data constants* the data value is shown.
- For *dense data vectors* the contents of the data vector is shown as list of numbers organized into rows and columns according to the indexes the data vector was defined over.
- For *sparse data vectors* the contents of the data vector is shown in a table with one line for each element in the vector.

For example to view the dense data vector *Demand* select the name in the *DATA* branch and press the *View* button. This will display the view window shown here below:



```
View Dense Data Vector: Demand
Demand[month,product] := (
  50000  24000  5000
  60000  30000  6000
  70000  36000  7000
  80000  42000  8000
  90000  48000  9000
  100000 52000  10000
  110000 50000  11000
  120000 48000  12000
  120000 44000  13000
  120000 40000  14000
  110000 36000  14000
  100000 32000  13000) ;
```

Figure 4.29: View Dense Data Vector Elements

The VARIABLES Branch

All defined variables for the current model are listed in the *VARIABLES* branch. If there are multiple *VARIABLES* sections in the model they will be shown as separate branches in the tree.

To obtain solution values, such as activity and reduced cost, for a variable, select the variable name in the tree and press the *View* button. This will open a view window showing the solution values for the selected variable. If the *Compute Ranges* option in the *General Solver Options* dialog box is *On* the sensitivity analysis ranges for the objective function coefficients are also shown. If the model has only been parsed, which means there are no solution values available, only the expanded name for each variable is shown.

The MACROS Branch

All defined macros for the current model are listed in the *MACROS* branch. If there are multiple *MACROS* sections in the model they will be shown as separate branches in the tree. To obtain more information about a macro, select the macro name in the tree and press the *View* button. This will open a view window containing the declaration and contents for the selected macro.

The CONSTRAINTS Branch

All defined constraints for the current model are listed in the *CONSTRAINTS* branch. To obtain solution values, such as slack and shadow prices, for a constraint, select the constraint name in the tree and press the *View* button. This will open a view window showing the solution values for the selected constraint.

VECTOR Capacity[month,machine] :

month	machine	Slack	Shadow Price
Jan	Grind	0.0000	-8.5714
Jan	Udrill	438.8571	0.0000
Jan	Hdrill	812.0000	0.0000
Jan	Boring	997.6286	0.0000
Jan	Planing	1109.0000	0.0000
Feb	Grind	766.0000	0.0000
Feb	Udrill	578.0000	0.0000
Feb	Hdrill	104.0000	0.0000
Feb	Boring	1060.0000	0.0000
Feb	Planing	1130.0000	0.0000
Mar	Grind	816.0000	0.0000
Mar	Udrill	558.0000	0.0000

Figure 4.30: View Constraint Solution Values

If the *Compute Ranges* option in the *General Solver Options* dialog box is *On* the sensitivity analysis ranges for the right-hand side are also shown. If the model has only been parsed, which means there are no solution values available, only the expanded name for each constraint is shown.

View Message Window

While MPL is running it can send various progress information to a message window. Choose Message Window from the View menu, to display the Message window. You can also press the View Message Window button in the Toolbar. If you want you can leave this window open while you are working in MPL as it will be updated automatically when you run your model.

What information is sent to the window will depend on which options are selected in the *Message Window* group of the *MPL Environment Options* dialog box.

- **Status Window Messages:** Sends all the status messages that are displayed in the topmost area of the *Status Window* to the *Message Window*.
- **MPL Input Lines:** Sends all input lines from the **MPL** model file to the *Message Window*.
- **Performance Statistics:** Sends performance statistics on parsing in the **MPL** model file to the *Message Window*.
- **Memory Usage Statistics:** Sends statistics for the **MPL** parser memory usage to the *Message Window*.
- **Warning Messages:** Sends all warning and error messages from **MPL** to the *Message Window*.
- **Database Connection:** Sends all messages concerning connection to databases to the *Message Window*.
- **SQL Statements:** Sends all SQL statements that are issued when importing and exporting data through the database connection to the *Message Window*.
- **Solver Iteration Log:** Sends all iteration log information from the solver to the *Message Window*.

Please refer to the section on the *MPL Environment Options* dialog box in *Chapter 4.9 The Options Menu* for more information.

4.8 The Graph Menu

The Graph menu is used when you want to display a graph of the matrix or of the objective function values for each iteration.



Figure 4.31: The Graph Menu

- | | |
|---------------------------|--|
| Matrix | - Display graph of the matrix |
| Objective Function | - Display graph of the objective function. |

Graph of the Matrix

To display a graph of the matrix nonzero elements, choose *Matrix* from the *Graph* menu. You can also press the *Graph Matrix* button in the Toolbar

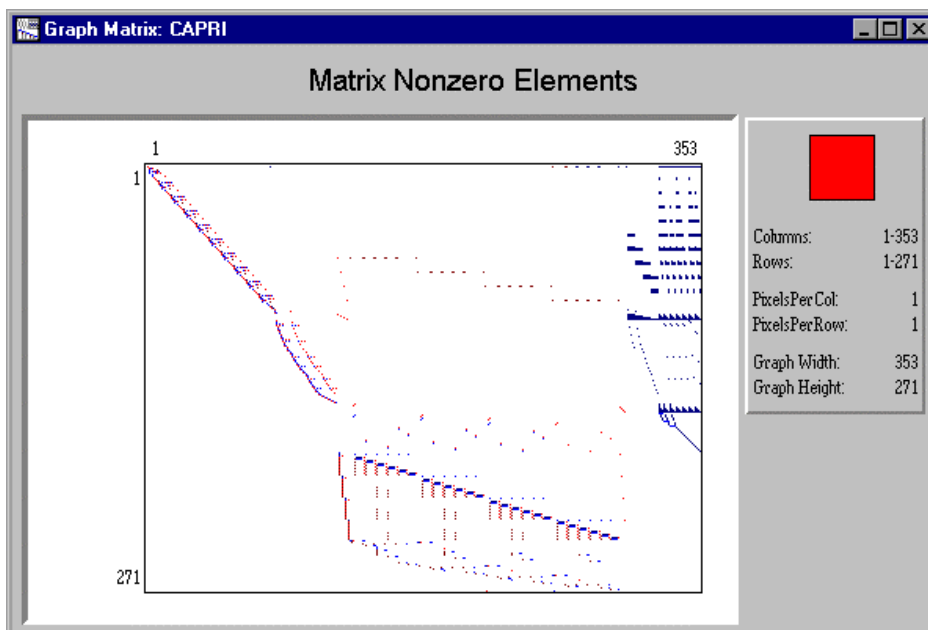


Figure 4.32: The Graph Matrix Window

Part II Using the MPL Modeling System

The graph shows all the positive numbers in blue and all the negative numbers in red. Furthermore, numbers that have binary values (1 and -1) are showed in lighter blue and lighter red respectively.

You can use the mouse to zoom into the matrix. Place the mouse cursor in the upper-left corner of the part of the matrix you want to zoom into. Hold the left mouse button down while you drag the mouse down and to the right. This will draw a box that shows the area that will be zoomed. When the box is the correct size release the left mouse button to zoom.

You can repeat the zoom several times until you reach the *spreadsheet view*. This will allow you to see the actual values and the names of each cell in the matrix. You can also reach the spreadsheet view by clicking with left mouse button on the part of the matrix you want to see.

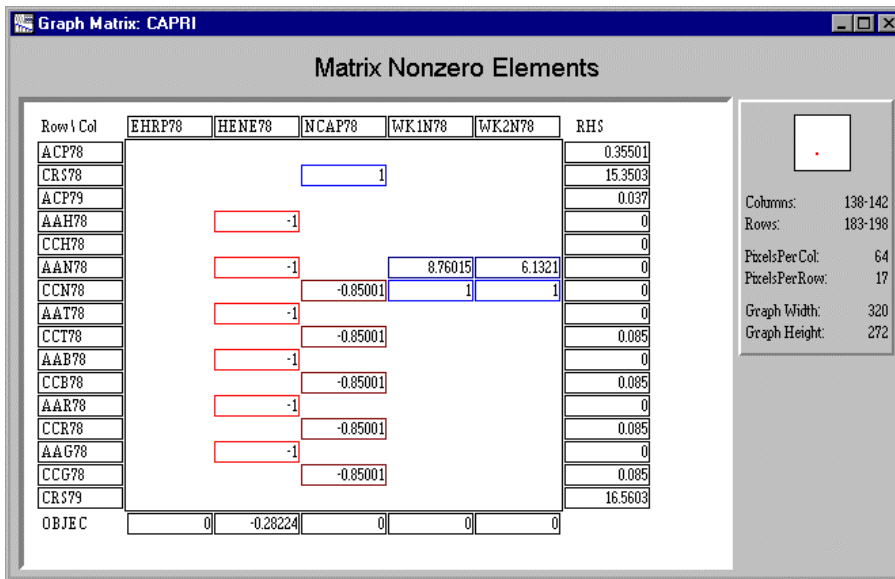


Figure 4. 33: The Spreadsheet View of Matrix

You can move around in the matrix using the arrow keys. The red dot in the small box in the upper right-hand corner shows you where you are located in the matrix. To zoom back out click the right mouse button.

Graph of the Objective Function

To display a graph of the objective function values for each iteration, choose *Objective Function* from the *Graph* menu. You can also press the *Graph Objective* button in the Toolbar. This will display a graph window shown below containing the objective function values.

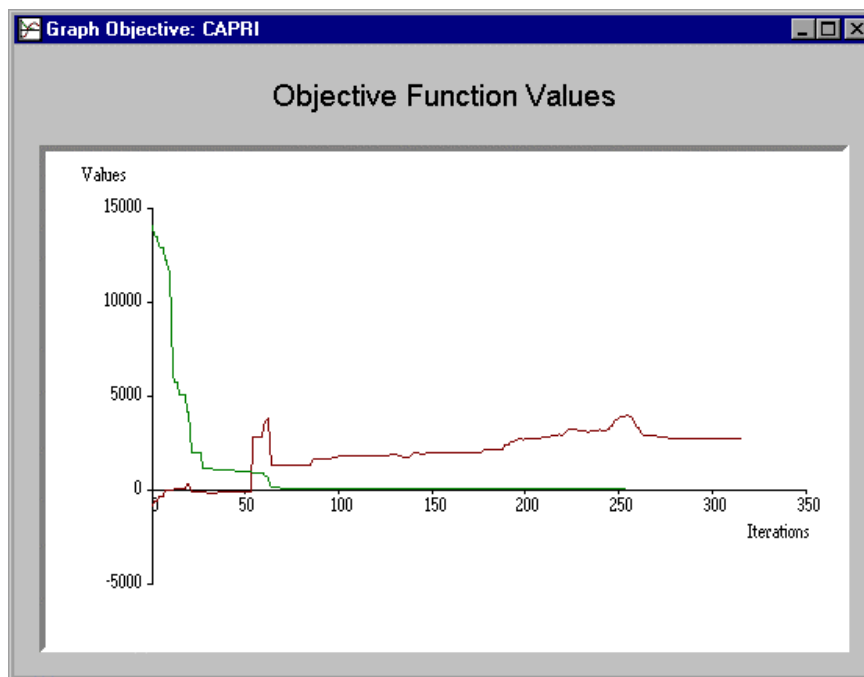


Figure 4.34: The Graph of Objective Function Window

The objective graph in **MPL** shows the objective function values in dark red. For iterations where there is no feasible solution the graph shows the amount of infeasibilities in green.

For integer problems the graph will also show the objective function value for each node (dark-blue) and the best integer solution value found so far (light-blue).

The graph of the objective function is only available when the model has been solved using Windows DLL solvers.

4.9 The Options Menu

The *Options* menu allows you to change various options for the solvers, the solution file, and for **MPL**.



Figure 4.35: The Options Menu

Environment...	- Change options for the MPL environment
MPL Language...	- Change options for the MPL language
Database...	- Change options for the database connection
Solution File...	- Change options for the solution file
Generate File...	- Change options for generated files
General Solver...	- Change general options for solvers
CPLEX parameters	- Change option parameters for CPLEX
XPRESS parameters...	- Change option parameters for XPRESS
Solver Parameters	- Change parameters for available solvers
Solver Option Lists...	- Change solver options in a property list format
Solver Menu...	- Sets up the menu of available solvers

Change MPL Environment Options

You can change the various preferences for **MPL** by choosing *Environment* from the *Options* menu. This will display the *Environment Options* dialog box shown below.

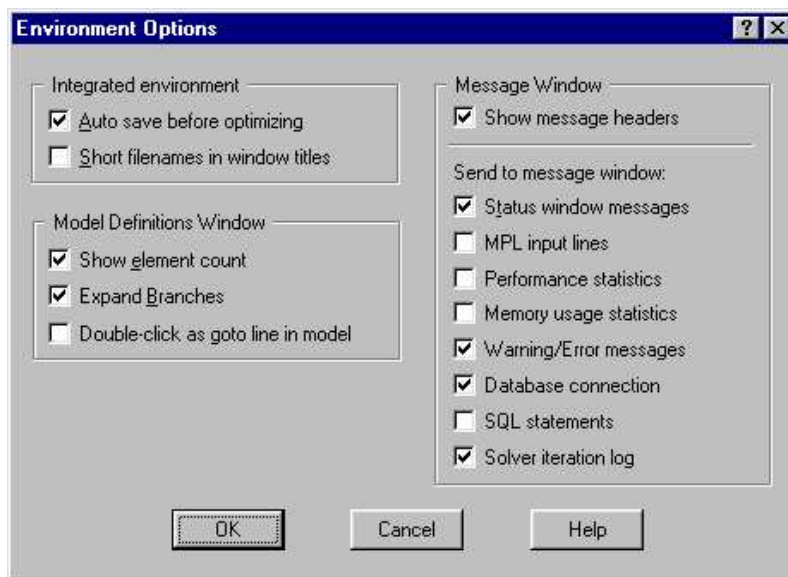


Figure 4.36: The Environment Options Dialog Box

Integrated environment

Auto Save: The model file is automatically saved whenever you optimize or parse the problem. This makes sure that you don't lose your work if something goes wrong during the optimization process.

Short filenames in window titles: **MPL** places the filename in the title bar of editor windows. This option directs whether the title bars for editor windows contain the full path information of the filename.

Part II Using the MPL Modeling System

Model definitions window

Show element count: Specifies whether the element count, for each item, is included in the *Model Definitions* window.

Expand Branches: Specifies whether each branch in the tree containing the model definitions is expanded. For larger models it may be beneficial to not expand the branches in order to be able to quickly find the branch you want to see.

Double-click as goto in model: If *On* double-clicking on a defined item in the tree brings the user to the line where the item was declared in the model. If *Off*, which is the default, double-clicking will open a view window with the defined item.

Message window

Show message headers: Specifies whether messages sent to the message window will be prefixed with a header text showing the type of the message.

Send to message window

Status window messages: Sends all the status messages that are displayed in the topmost area of the *Status Window* to the *Message Window*.

MPL input lines: Sends all input lines from the **MPL** model file to the *Message Window*.

Performance statistics: Sends performance statistics on parsing in the **MPL** model file to the *Message Window*.

Memory usage statistics: Sends statistics on the **MPL** parser memory usage to the *Message Window*.

Warning messages: Sends all warning and error messages from **MPL** to the *Message Window*.

Database Connection: Sends all messages concerning connection to databases to the *Message Window*.

SQL statements: Sends all SQL statements that are issued when importing and exporting data through the database connection to the *Message Window*.

Solver Iteration Log: Sends all iteration log information from the solver to the *Message Window*. This is the same option as the *Send iteration log to message window* option in the *General Solver Options* dialog box.

Change MPL Language Options

You can change the various preferences for **MPL** by choosing *MPL Language* from the *Options* menu. This will display the *MPL Language Options* dialog box shown below.

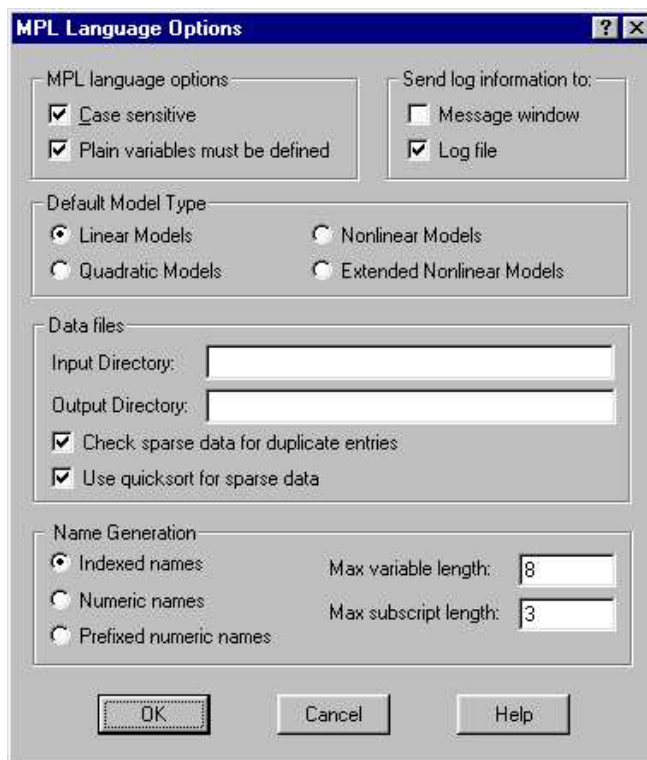


Figure 4.37: The MPL Language Options Dialog Box

MPL language options

Case sensitive: Directs whether the **MPL** language is case sensitive or not. The default is case sensitive *On*.

Plain variables must be defined: Specifies whether plain variables must be defined in the Decision Variables section of the model or can be introduced as they appear in the model. For larger models, requiring plain variables to be defined, can increase the maintainability of the model.

Part II Using the MPL Modeling System

Send log information to:

Message window: Specifies whether log information from the **MPL** language parser are send to the *Message Window*. This is the same option as the *Performance statistics* option in the *MPL Environment Options* dialog box.

Log file: Specifies whether log information from the **MPL** language parser are to send to file.

Default model type

- Linear Models Specifies that the **MPL** language parser should only accept models that are either linear or mixed integer.
- Quadratic Models Specifies that the **MPL** language parser should accept models that are quadratic in addition to the standard linear or mixed integer models. Please note, that in order to solve quadratic problems, you will need a optimizer that supports quadratic programming, such as the CPLEX Barrier solver.
- Quadratic problems are defined to have the objective function of the following form: $Q * x^T * x + c^T x$ where T means that the vector is transposed. Q is a matrix of quadratic objective function coefficients. The elements Q_{jj} are coefficients of the quadratic terms x_j^2 while the elements Q_{ij} and Q_{ji} are summed together to be the coefficient of the term $x_i * x_j$.
- There are two different types of Q matrices that are supported: a separable problem is one where only the diagonal terms of the matrix are defined, and a non-separable is one where at least one off-diagonal term of the matrix is zero. Quadratic solvers, such as CPLEX Barrier, will solve only convex quadratic minimization problems (or concave maximization problems). For convex problems, Q must be positive semi-definite, which means that the term $Q * x^T * x \geq 0$ for any x, whether or not x is feasible. (For maximization problems, the requirement is that Q is negative semi-definite, which means that $Q * x^T * x \leq 0$ for all x). Note that for separable Q (in a minimization problem), Q positive semi-definite is equivalent to $Q \geq 0$.
- Nonlinear Models Specifies that the **MPL** language parser should accept models that are nonlinear, for example have variables multiplied together or uses one of the arithmetic functions like LOG or EXP on variables. Please note, that in order to solve nonlinear problems, you will need a solver that handles nonlinear models. **MPL** currently supports the nonlinear solvers CONOPT from ARKI Consulting and LSGRG2 from Optimal Methods.

Data files

Input Directory: Selects the folder where **MPL** will search for input data files. The default is the current folder.

Output Directory: Selects the folder where **MPL** will save output data files. The default is the current folder.

Check Sparse Data For Duplicate Entries: Specifies whether sparse data files are checked for duplicate index entries. In some cases the user may want to read in a data file without receiving errors even if it has duplicate entries. When there are duplicate entries, the last entry in the file will be used by **MPL**.

Use quicksort for sparse data: Specifies whether sparse data is sorted with *quicksort* after it is read in. It is normally faster to use *quicksort* but if there is an invalid entry, such as duplicates, in the data file **MPL** will not be able to pinpoint the problem line accurately.

Name generation

Indexed names: Directs **MPL** to generate names for vector variables and constraints using the indexes they were defined with.

Numeric names: Directs **MPL** to generate variable names as numeric with the prefix 'C' and constraint names with the prefix 'R'.

Prefixed numeric names: Directs **MPL** to generate names for vector variables and constraints as numeric with the prefix based on the vector name.

Max variable length: Most LP solvers have a restriction on the length of variable names. Since **MPL**, in most cases, sends the matrix directly through memory to the solver, the variable names are normally not needed. If they are needed, the value set here helps you ensure that the variable names generated are within limits. The default value is 8.

Max subscript length: This value decides how many characters of indexes are retained in the generated variable name. This allows you to use long index names in the model, but keep variable names concise in the generated input file. The default value is 3.

Database Options

You can change the various preferences for **MPL** by choosing *Database* from the *Options* menu. This will display the *Database Options* dialog box shown below.

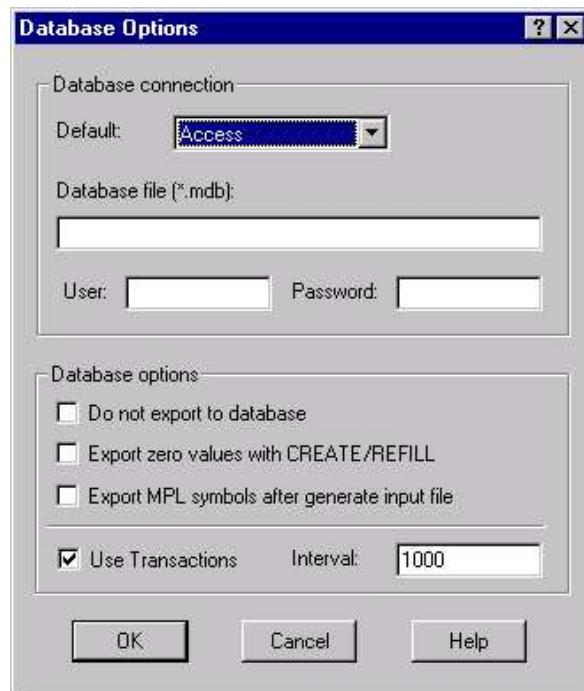


Figure 4.38: The Database Options Dialog Box

Database connection

Default: Chooses which of the supported databases is the default for the *MPL Database Connection*. **MPL** supports both specific databases, such as *FoxPro* and *Paradox*, as well as the databases that are supported by the *ODBC* standard such as *Access*.

Directory or data source

Depending on which database is selected, this entry will specify the database directory or the data source that **MPL** will use as default when importing data. For Xbase type databases, such as *FoxPro* and *Paradox*, this will typically be the directory where the database files are stored. For *ODBC* type databases this either be a database file, such as *<filename.mdb>* for *MS Access* and *<filename.xls>* for *Excel*, or a defined *ODBC* data source that can be specified in the *ODBC* control panel.

Username

When you are working with databases that require a Username to log in, this option can be used to specify it.

Password

When you are working with databases that require a Password to log in, this option can be used to specify it.

Database options

Do not export: Do not export solution values to database even if there are export statements in the model.

Export zero values: When exporting using either *CREATE* or *REFILL*, this option specifies whether records with zero activity values should also be exported. The default value is *Off* since in most cases only the nonzero records are needed.

Export MPL symbols: When exporting **MPL** symbols such as; indexes and datavectors, they are generally only exported when model is solved and not when an input file is generated. This options directs **MPL** to export symbols when input files are generated, since this might be useful for debugging purposes. The default value is *Off*.

Use Transactions: Directs **MPL** to use transactions when exporting to a database. Using transactions can greatly improve speed especially when working with remote databases. Please note, that not all databases support transactions. The default value is *On*.

Interval: Specifies what the interval should be between when the transactions are performed.

Change Solution File Options

You can change the contents and various options for the solution file by choosing *Solution File* from the *Options* menu. This will display the *Solutions File Options* dialog box shown below.

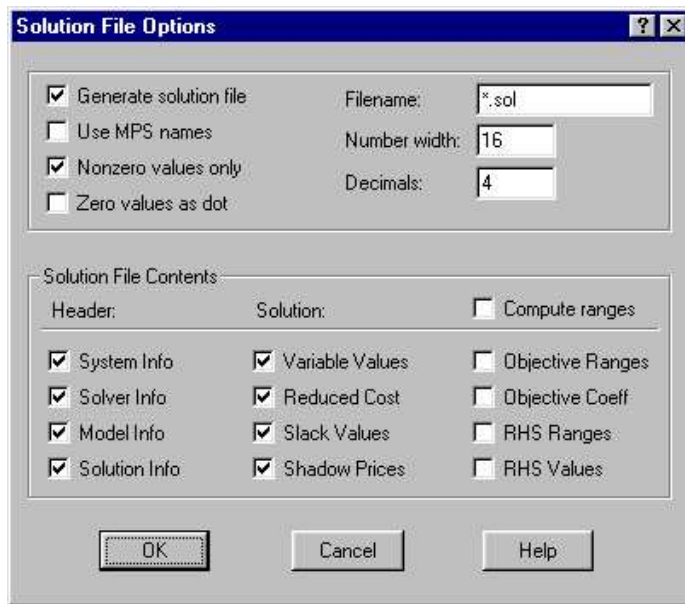


Figure 4.39: Solution File Options Dialog Box

Here is a list of the options that you can change with explanations to follow.

Generate Solution File: Directs whether **MPL** creates a solution file automatically after solving the problem.

Use MPS Names: Directs whether the solution file uses MPS type names for constraints and variables instead of listing the value of each subscript in columns.

Nonzeros values only: Directs whether the solution should list all solution values or nonzero values only.

Zero Values as Dot: Directs whether zero values in the solution are listed using a dot instead of zero.

Filename: Specifies the filename **MPL** will use to save the solution file. If the filename given contains star (*) instead of the name, like *.sol, **MPL** will use the name of model file with the extension given.

Number width: Set the field size for number values in the solution file.

Decimals: Set the number of places after the decimal point.

Solution file contents

System Info: Includes various statistics from the system and about the problem in the solution file; filename, date and time when run, and how much time the parsing of the model took.

Solver Info: Includes various statistics about the optimization process in the solution file; which solver was used, value of the objective function, number of iterations, number of nodes for integer problems, and how much time the optimization took.

Model Info: Includes various statistics about the model in the solution file; the problem name, number of constraints, number of variables, number of nonzeros, density of the matrix, and number of integer variables for integer problems.

Solution Info: Includes various statistics about the solution result in the solution file; status of the solution reported from the solver, and then the sense, name, and value of the objective function.

Variable Values: Includes the activity values for each variable.

Reduced Cost: Includes the reduced costs for each variable.

Slack Values: Includes the slack values for each constraint.

Shadow Prices: Includes the shadow prices for each constraint.

Compute Ranges: Directs whether solution ranges are retrieved from the solver and stored in **MPL** memory. This option is automatically selected if either objective or RHS ranges are selected for the solution file contents. This is the same option as the *Compute Ranges for Sensitivity Analysis* option in the *General Solver Options* dialog box.

Objective Ranges: Includes the objective function ranges for each variable.

Objective Coeff: Includes the objective function coefficient for each variable.

RHS Ranges: Includes the RHS ranges for each constraint.

RHS Values: Includes the RHS value for each constraint.

Change Generate File Options

You can change various options for generated files by choosing *Generate File* from the *Options* menu. This will display the *Generate File Options* dialog box shown below.

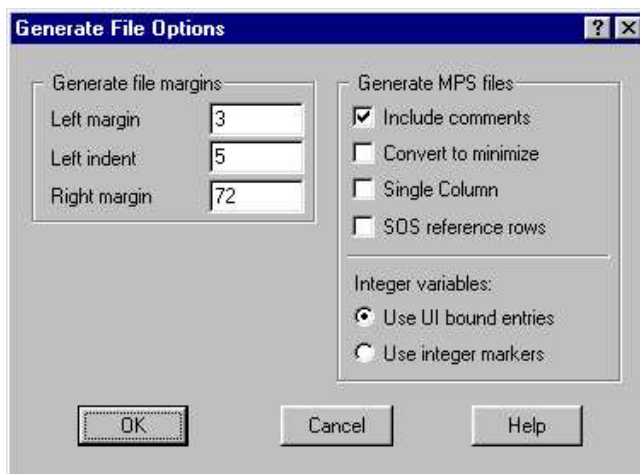


Figure 4.40: Generate File Options Dialog Box

Generate file margins

Left Margin: Sets the left margin for the generated input file. The default value is 3.

Left Indent: Sets the left indent for the generated input file. The default value is 5.

Right Margin: Sets the right margin for the generated input file. The default value is 72.

Generate MPS files

Include comments: This option determines whether **MPL** should place comments in the header of the MPS file that give various statistics about the problem. The default is *On*.

Convert to minimize: This option determines whether the sense of the objective function will be converted to minimize regardless of how the original formulation was set up. This can be useful when creating MPS files for packages that require MIN problems.

SOS reference rows: This option determines whether reference rows can be specified when creating SOS marker records. When this option is set the *SETORG* entry will be moved to field 4 to make space for the reference row which will be put in field 6.

Single column: This option determines whether the MPS file is written with one column of data in the *COLUMN* section. The default is *Off* for two columns of data.

Integer variables in generate MPS files

Use UI bound entries: This option determines whether integer variables will be specified using *UI* records in the *BOUNDS* section in the generated MPS file. This is the default.

Use integer markers: This option determines whether integer variables will be specified using integer marker records in the *COLUMNS* section in the generated MPS file.

Change General Solver Options

You can change various general options for solvers by choosing *General Solver* from the *Options* menu. This will display the *General Solver Options* dialog box shown below.

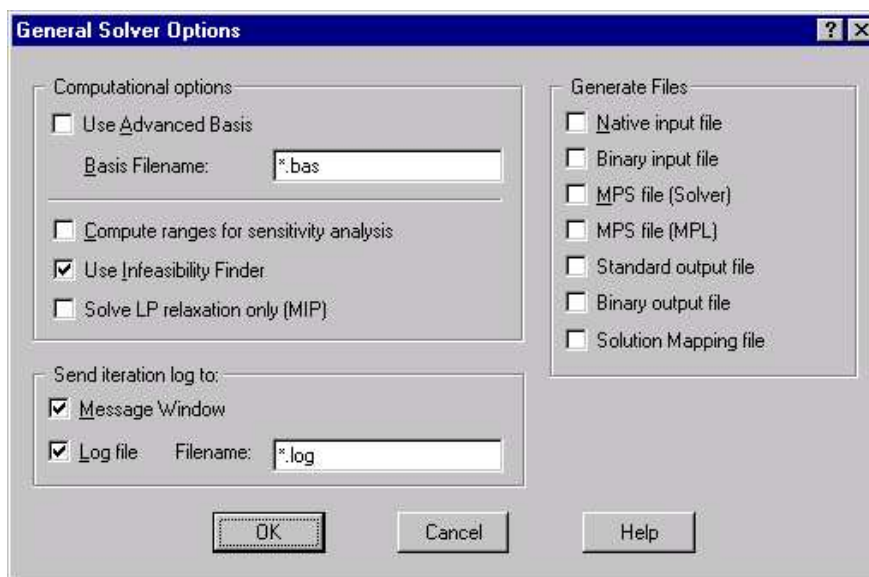


Figure 4.41: The General Solver Options Dialog Box

Computational options

Use Advance Basis: The advance basis for the previous optimization is used as the starting point for the next optimization. The default is *On*.

Basis Filename: Specifies the filename the solver will use for the basis file. If the filename given contains asterisks '*' instead of the name, like the default entry '*.bas', the solver will use the name of model file with the extension given.

Compute ranges for sensitivity analysis

Directs whether solution ranges are retrieved from the solver and stored in **MPL** memory. This option is automatically selected if either objective or RHS ranges are selected for the solution file contents.

Use infeasibility finder

If the model being solved is infeasible, solvers such as CPLEX can automatically invoke an infeasibility finder to help locate the problem. In some cases the infeasibility finder can not help or takes a very long time. This option can then be used to turn off the infeasibility finder.

Solve LP relaxation only (MIP)

Directs MPL to not send any information about integer variables to the solver and solve only the LP relaxation for the model.

Send iteration log to:

Message window: While the solver is optimizing the iteration log information will be sent to the *Message Window*.

Log file: While the solver is optimizing the iteration log information will be sent to a log file.

Log filename: Specifies the filename the solver will use for the log file. If the filename given contains asterisks '*' instead of the name, like the default entry '*.log' the solver will use the name of model file with the extension given.

Generate files

Native input file: Directs the solver to write out its native input file for the matrix it received from **MPL**. This option can be very useful for debugging purposes.

Binary input file: Directs the solver to write out its binary input or save file for the matrix it received from **MPL**.

MPS file: Directs the solver to write out a MPS file for the matrix it received from **MPL**.

MPS file (MPL): Directs the **MPL** to write out a MPS file for the matrix it generated. This option can be very useful for debugging purposes.

Standard output file: Directs the solver to write its standard output file for the solution.

Binary output file: Directs the solver to write out a binary output file for the solution.

Solution mapping file: Directs **MPL** to write out a mapping file that shows how variables and constraints sent to the solver are mapped on to the **MPL** vector variables and constraints. This can be useful when writing a program that will read the output files generated by **MPL**.

Change CPLEX Simplex Options

You can change the simplex options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *Simplex* tab. This will display the simplex options for CPLEX in the dialog box as shown below:

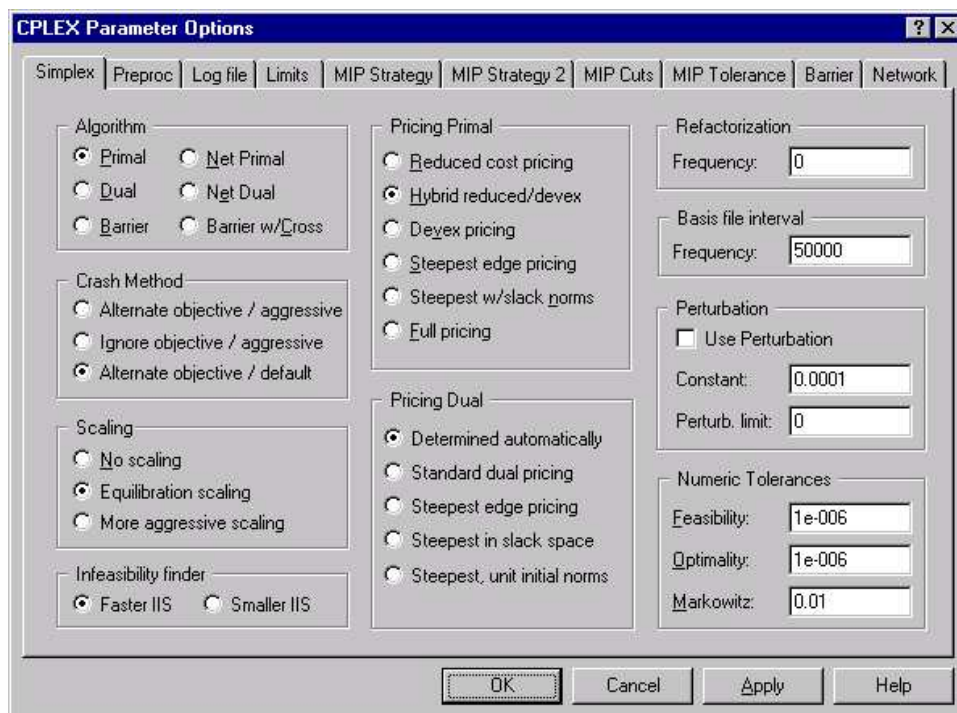


Figure 4.42: The Simplex Tab in CPLEX Options Dialog Box

Algorithm

The algorithm option selects which algorithm to use when optimizing the LP problem. Primal simplex is the default algorithm, but always try to dual simplex as well, particularly for difficult problems. Barrier works well on many large problems. The network algorithm is fastest if a large network structure exists in the problem.

<i>Primal</i>	Use the primal simplex algorithm (default).
<i>Dual</i>	Use the dual simplex algorithm.
<i>Barrier</i>	Use the barrier algorithm.
<i>Net Primal</i>	Use the network algorithm and then primal simplex.
<i>Net Dual</i>	Use the network algorithm and then dual simplex.
<i>Barrier w/Cross</i>	Use the barrier algorithm followed by an automatic crossover to a basic solution.

Part II Using the MPL Modeling System

Crash method

The Crash method option determines the way CPLEX orders variables relative to the objective function when selecting an initial basis. Standard order usually works well, but experimentation is required to determine if changing this option will benefit the problem solution efficiency.

<i>Alternate objective/aggressive</i>	If primal, alternate ways of using objective coefficients; else, if dual aggressive starting basis.
<i>Ignore objective/aggressive</i>	If primal, ignore object coefficients during crash; else, if dual aggressive starting basis.
<i>Alternate objective/default</i>	If primal, alternate ways of using objective coefficients; else, if dual default starting basis (default).

Scaling

The Scaling option determines the scaling of the problem matrix. If your problem has difficulty remaining feasible during the solution process, try aggressive scaling.

<i>No scaling</i>	No scaling is to be done.
<i>Equilibration scaling</i>	Use the equilibration scaling method (default).
<i>More aggressive scaling</i>	Use a modified, more aggressive scaling method that can produce improvements on some problems.

Infeasibility finder

The Infeasibility finder option determines the method to be used to identify the IIS set for an infeasible model.

<i>Faster IIS</i>	Faster method that works best for most models (default).
<i>Smaller IIS</i>	This method produces a smaller IIS set, but more computation time is usually needed.

Pricing primal

The Pricing Primal option selects the pricing algorithm for the primal simplex. The default hybrid pricing usually provides the fastest solution time but many problems can benefit from alternate settings.

<i>Reduced cost pricing</i>	Use reduced-cost pricing.
<i>Hybrid reduced/devex</i>	Use Hybrid reduced-cost and Devex pricing (default).
<i>Devex pricing</i>	Use Devex pricing.
<i>Steepest edge pricing</i>	Use Steepest-edge pricing.
<i>Steepest w/slack norms</i>	Use Steepest-edge pricing with slack initial norms.
<i>Full pricing</i>	Use Full pricing.

Pricing dual

The Pricing Dual option selects the pricing algorithm for the dual simplex. While the default pricing usually provides the fastest solution time, many problems benefit from alternate settings.

<i>Determined automatically</i>	Pricing determined automatically (default).
<i>Standard dual pricing</i>	Use Standard dual pricing.
<i>Steepest edge pricing</i>	Use Steepest-edge pricing.
<i>Steepest in slack space</i>	Use Steepest-edge pricing in slack space.
<i>Steepest, unit init norms</i>	Use Steepest-edge pricing, unit init norms.

Refactorization

The number of iterations between refactorizations of the basis matrix. In the default setting of zero, CPLEX determines a setting automatically at run time.

Basis interval

The number of iterations between refactorizations of the basis matrix. In the default setting of zero, CPLEX determines a setting automatically at run time.

Use perturbation

When the perturbation option is *On* all problems will automatically be perturbed as optimization begins. When *Off* allows CPLEX to determine dynamically, during solution, whether progress is slow enough to merit a perturbation. The default is *Off*, the situations in which a perturbation helps will be rare and restricted to problems that exhibit extreme degeneracy.

Perturbation constant

The perturbation constant sets the amount by which CPLEX will perturb the upper and lower bounds on the variables when a problem is perturbed. This parameter can be set to a smaller value if the default value creates too large a change in the problem.

Perturbation limit

The primal and dual simplex methods include a perturbation mechanism for dealing with situations in which no progress has been made in the objective function over a significant number of iterations. This phenomenon is sometimes called stalling. With default settings, the number of stalled iterations before perturbation is invoked is determined internally by CPLEX depending upon problem dimensions. However, when the parameter is set to a positive value by the user, that value becomes the limit on stalled iterations before perturbation will be performed.

Feasibility tolerance

The feasibility tolerance specifies the degree to which a problem's basic variables may violate their bounds. This tolerance influences the selection of an optimal basis and can be reset to a lower value when a problem is having difficulty maintaining feasibility during optimization. Default value is 1e-06.

Optimality tolerance

The optimality tolerance specifies the reduced cost tolerance for optimality. This option governs how closely CPLEX must approach the theoretically optimal solution. Default value is 1e-06.

Markowitz tolerance

The Markowitz tolerance influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution. Default value is 0.01.

Change CPLEX Preprocessing Options

You can change the preprocessing options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *Preproc* tab. This will display the preprocessing options for CPLEX in the dialog box as shown below:

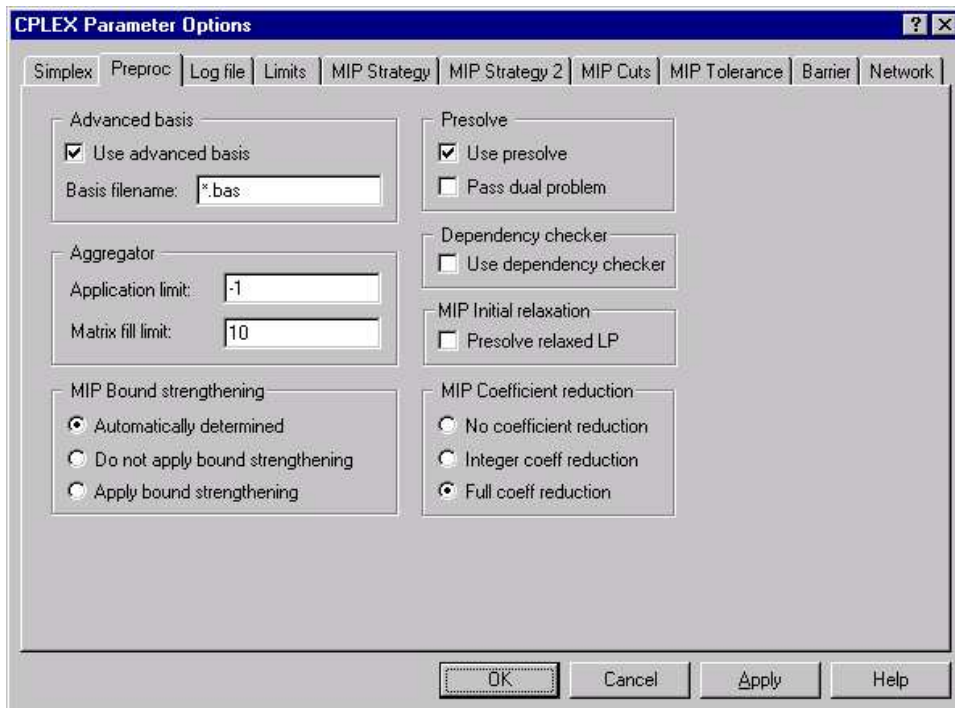


Figure 4.43: The Preproc Tab in CPLEX Options Dialog Box

Advanced basis

Use advanced basis: The advance basis for the previous optimization is used as the starting point for the next optimization. The default is *On*.

Basis Filename: Specifies the filename CPLEX will use for the basis file. If the filename given contains asterisks '*' instead of the name, like the default *.bas CPLEX will use the name of model file with the extension given.

Application limit

The aggregator, when set to a nonzero value, will invoke the CPLEX Aggregator to use substitution where possible to reduce the number of rows and columns before the problem is solved. If the parameter is set to a positive value then, the aggregator will be applied the specified number of times, or until no more reductions are possible. At the default value of -1, the aggregator is applied once for linear programs, and an unlimited number of times for mixed integer problems. At this setting, all potential presolve reductions are performed for mixed integer programs.

Matrix fill limit

Sets the Matrix fill limit for the aggregator. By default, determined automatically. This parameter should only rarely require adjustment.

MIP bound strengthening

The Bound Strengthening option is used when solving mixed integer programs. Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during branch-and-bound. This reduction is usually beneficial, but occasionally, due to its iterative nature, takes a long time.

Automatically determined

Do not apply bound strengthening

Apply bound strengthening

Use presolve

The Presolve option, when set to *On*, will invoke the CPLEX Presolve to make problem simplifications and reductions.

Pass dual problem

The Pass dual problem option determines if CPLEX Presolve should pass the primal or dual linear programming problem to the linear programming optimization algorithm. By default, CPLEX Presolve is applied to the primal problem, and the resulting primal problem is passed to the optimizer. If the Pass Dual Problem option is set to *On*, then the CPLEX presolve algorithm will still be applied to the primal problem, but the resulting dual linear program is passed to the optimizer. This is a useful technique for problems with more constraints than variables.

Dependency checker

The dependency checker option determines if the “dependency checker” is activated. If *On*, the dependency checker will search for dependent rows during preprocessing. If *Off* (default), dependent rows will not be identified. For many models, eliminating the dependency check will speed up the preprocessing time, at the expense of not identifying dependent rows.

Presolve relaxed LP

When the Presolve Relaxed LP option is *On*, CPLEX will invoke the Presolve for linear programs for the initial relaxation of a mixed integer program, according to the other CPLEX Presolve parameter settings. Sometimes additional reductions can be made beyond any MIP presolve reductions that may have already been done.

MIP coefficient reduction

Coefficient reduction is a technique used when presolving mixed integer programs. The benefit of coefficient reduction is to improve the objective value of the initial (and subsequent) linear programming relaxations solved during branch-and-bound by reducing the number of non-integral vertices. However, the linear programs generated at each node may become more difficult to solve. There is a resulting tradeoff between reducing the number of nodes in the branch-and-bound tree and the time to solve each node via a linear programming algorithm. Full coefficient reduction reduces all possible coefficients, while integer coefficient reduction will only reduce coefficients to integer values.

<i>No Coefficient Reduction</i>	No coefficient reduction.
<i>Integer Coeff. Reduction</i>	Reduce only to integral coefficients.
<i>Full Coeff. Reduction</i>	Reduce all possible coefficients. (default)

Change CPLEX Log File Options

You can change the log file options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *Log file* tab. This will display the log file options for CPLEX in the dialog box as shown below:

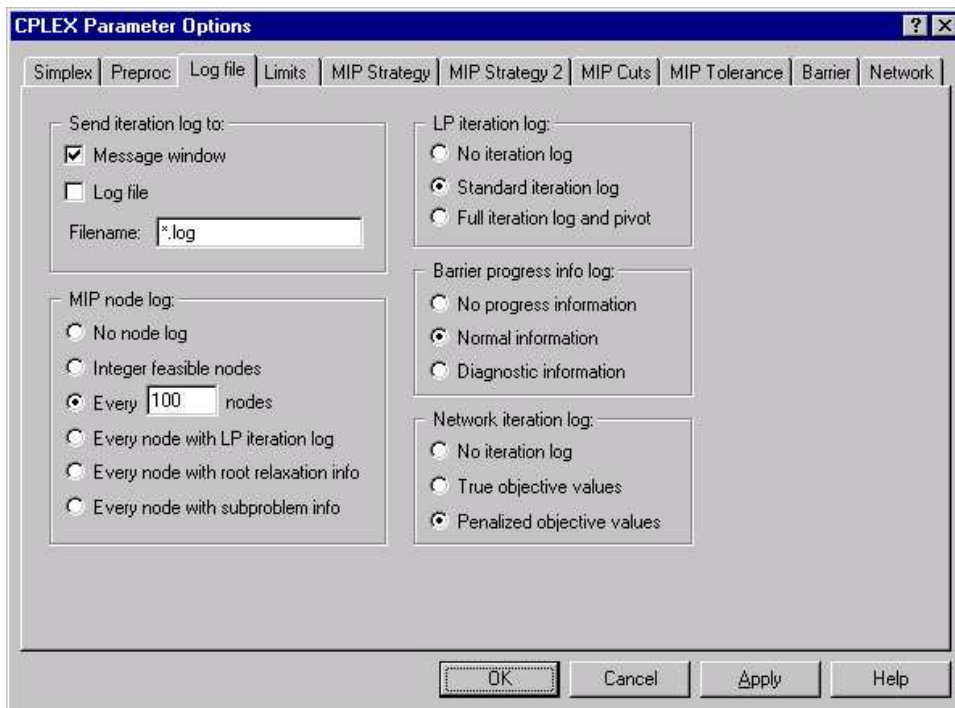


Figure 4.44: The Log file Tab in CPLEX Options Dialog Box

Send iteration log to:

Message window: While CPLEX is optimizing the iteration log information will be sent to the **MPL** Message Window. This is the same option as the *Send iteration log to message window* option in the *General Solver Options* dialog box.

Log file: While CPLEX is optimizing the iteration log information will be sent to a log file. This is the same option as the *Send iteration log to log file* option in the *General Solver Options* dialog box.

Log filename: Specifies the filename CPLEX will use for the log file. If the filename given contains asterisks ***** instead of the name, like the default **.log*, CPLEX will use the name of model file with the extension given.

Part II Using the MPL Modeling System

MIP node log

The MIP Node log option controls the frequency of the node logging. This option is useful for monitoring the progress of a problem that requires many nodes to solve.

<i>No node log</i>	No node information appears in the log.
<i>Feasible nodes</i>	Node information appears only for each integer feasible solutions found
<i>Every n-th node</i>	Node information appears for every n -th node where n is the number given in the input box. The default is every 100 th nodes.
<i>Every node and iteration</i>	Node information appears for every node as well as the standard LP iteration log.
<i>Every node and relaxation</i>	Node information appears for every node as well as the LP root relaxation info.
<i>Every node and subproblem</i>	Node information appears for every node as well as the LP subproblem info.

LP iteration log

The LP iteration log controls the amount of information that is sent to either the log window and/or the log file.

<i>No Iteration log</i>	No iteration information appears in the log.
<i>Standard iteration log</i>	Iteration information appears for every refactorization.
<i>Full iteration log and pivot</i>	Iteration information appears for every iteration.

Barrier progress info log

The Barrier progress info log option controls the level of progress information to be reported.

<i>No progress information</i>	No progress information appears in log.
<i>Normal information</i>	Normal progress information appears in log.
<i>Diagnostics information</i>	Diagnostic information appears in log.

Network iteration log

<i>No iteration log</i>	No iteration information appears in the log.
<i>True objective values</i>	True objective values are included in the log.
<i>Penalized objective values</i>	Objective values, including monotonic values are included in the log.

Change CPLEX Limit Options

You can change the limit options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *Limits* tab. This will display the limit options for CPLEX in the dialog box as shown below:

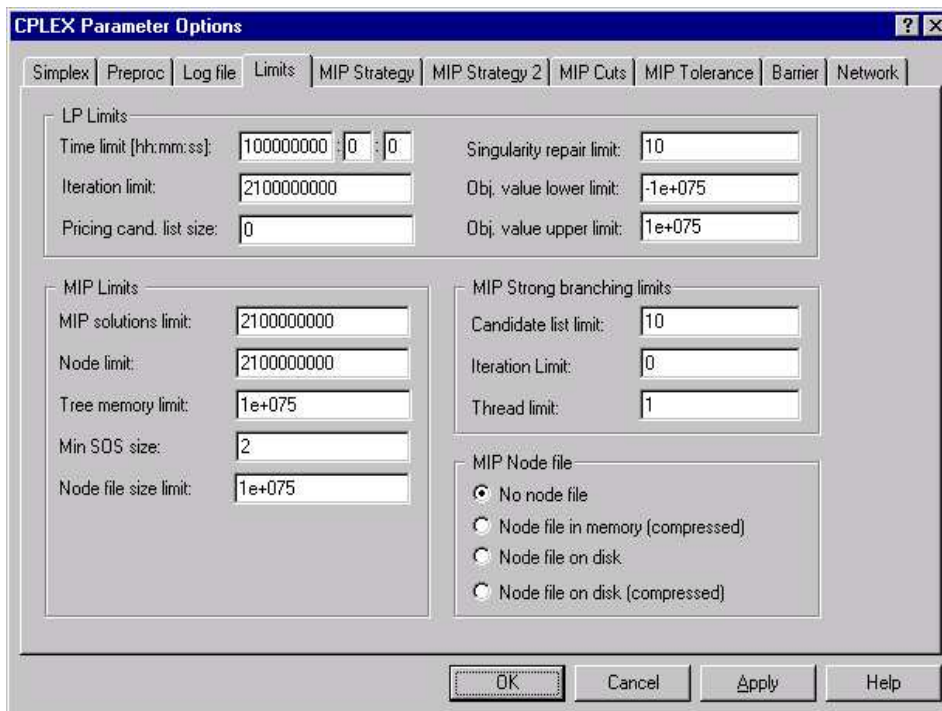


Figure 4.45: The Limits Tab in CPLEX Options Dialog Box

Time limit

The Time limit option used to set maximum time (in hours:min:sec) for computations before termination. The time limit applies to primal simplex, dual simplex, barrier, and mixed integer optimizations, as well as infeasibility finder computations.

Iteration limit

The Iteration limit sets the maximum number of iterations before the algorithm terminates, without reaching optimality. Be sure to enter only an integer value. The default value is 2100000000.

Pricing candidate list size

Contains the maximum number of variables kept in the pricing candidate list. If the value is zero the number is determined automatically at run-time.

Singularity limit

The Singularity limit value restricts the number of times CPLEX will attempt to repair the basis when singularities are encountered. Once the limit is exceeded, CPLEX replaces the current basis with the best factorizable basis that has been found. At this point the user can examine and modify the problem or increase the singular limit to force CPLEX to work harder at eliminating singularities. The default value is 10.

Object value lower limit

Setting a lower objective function limit will cause CPLEX to halt the optimization process once the minimum objective function value limit has been exceeded. This limit applies only during Phase II of the optimization.

Object value upper limit

Setting an upper objective function limit will cause CPLEX to halt the optimization process once the maximum objective function value limit has been exceeded. This limit applies only during Phase II of the optimization.

MIP solutions limit

The MIP solutions limit value limits the mixed integer optimization to finding only this number of mixed integer solutions before stopping. The default value is 2100000000.

Node limit

The Node limit value sets the maximum number of nodes solved before the algorithm terminates, without reaching optimality. The default value is 2100000000.

Tree memory limit

The Tree memory limit value sets an upper limit on the amount of memory (in megabytes) that the branch-and-bound tree can consume. CPLEX will terminate optimization when the amount of memory required to store branch-and-bound information exceeds the tree memory parameter setting.

Min SOS size

The Min SOS Size parameter is used to set the minimum size for sets found during the scan for SOS Type 3 sets. The SOS algorithm may not be worthwhile on smaller size sets. The default value is 2.

Node file size limit

The Node File Size Limit option limits the size of the node file.

MIP strong branching limits

- Candidate list limit* Controls the length of the candidate list when using the “strong branching” variable selection setting. Default is 10.
- Iteration list limit* Controls the number of simplex iterations performed on each variable in the candidate list when using the “strong branching” variable selection setting. The default setting 0 chooses the iteration limit automatically.
- Thread limit* Controls the number of parallel threads used to perform strong branching. This parameter does nothing if the MIP thread limit is greater than 1

MIP node file

The MIP Node File is used when the tree memory limit is reached. If no node file is selected the optimization is terminated when the tree memory limit is reached. Otherwise, a group of nodes is removed from the in-memory set, and transferred to a node file. This group of nodes is returned to the in-memory set as needed.

Node files are used most efficiently when the amount of tree memory is reasonably large, so that the node files do not have to be formed too frequently. The compression options require a small amount of extra time due to the extra processing but since they result in smaller node files the overall systems is smaller and the system throughput may increase.

- No Node File* No node file (default).
- Node File in Memory (Compressed)* Node file in memory and compressed.
- Node File on Disk* Node file on disk.
- Node File on Disk (Compressed)* Node file on disk and compressed.

Change CPLEX MIP Strategy Options

You can change the MIP strategy options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *MIP Strategy* tab. This will display the MIP strategy options for CPLEX in the dialog box as shown below:

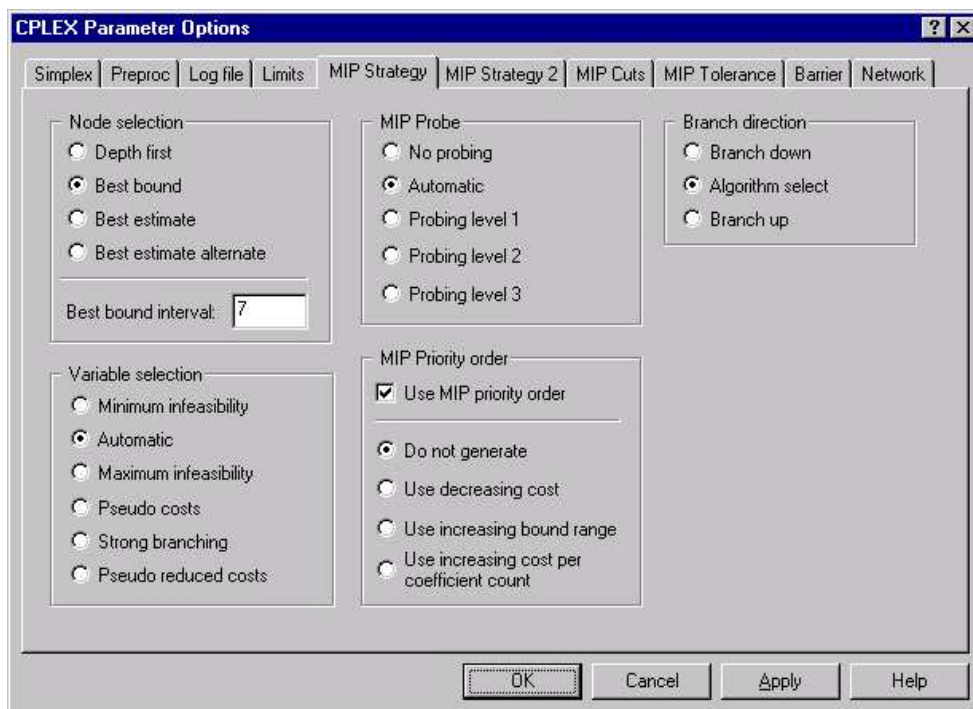


Figure 4.46: The MIP Strategy Tab in CPLEX Options Dialog Box

Node selection

The Node selection option is used to set the rule for selecting the next node to process when backtracking (proceeding back through the tree when a node is infeasible or cutoff). The Best Bound method works better for many problems; but Depth First may find a good solution deep in the tree faster. Depth first will also use less memory.

Depth first The depth-first search strategy chooses the most recently created node.

Best bound The best bound strategy chooses the node with the best objective function for the associated LP relaxation (default).

Best estimate The best estimate strategy selects the node with the best estimate of the integer objective value that would be obtained from a node once all integer infeasibilities are removed.

Best estimate Alternate Alternate best-estimate search.

Best bound interval

When using Best-Estimate Search, the Best Bound Interval is the interval at which the best bound node, instead of the best estimate node, will be selected from the tree.

- 0 Best estimate node always selected
- 1 Best bound node always selected
- >1 The interval at which the best bound node will be selected from the tree.

The default value is 7.

Variable selection

The Variable selection option is used to set the rule for selecting the branching variable at the node which has been selected for branching.

- Minimum infeasibility* The minimum infeasibility rule chooses the variable with the smallest fractional value. It may lead more quickly to a first integer feasible solution, but will usually be slower overall to reach the optimal integer solution.
- Automatic* Branch variable automatically selected. This option allows CPLEX to select the best rule based on the problem and its progress (default).
- Maximum infeasibility* The maximum infeasibility rule chooses the variable with the largest fractional value; It forces larger changes earlier in the tree, which tends to produce faster overall times to reach the optimal solution.
- Pseudo costs* Pseudo costs causes variable selection based on pseudo-costs which are derived from pseudo-shadow prices.
- Strong branching* Strong branching causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is the most promising. This strategy can be effective on large, difficult MIP problems
- Pseudo reduced costs* Pseudo-reduced costs causes variable selection based on pseudo-costs which are derived from pseudo-shadow prices.

MIP probe

Determines the amount of variable probing to be performed on a problem. Probing can be both very powerful and very time consuming. Setting the value to *Probing level 1-3* can in some cases result in dramatic reductions or dramatic increases in solution time on particular models.

- No probing*
- Automatic*
- Probing level 1*
- Probing level 2*
- Probing level 3*

Part II Using the MPL Modeling System

MIP priority order

A priority order assigns a branching priority to some or all of the integer variables. Variables with priorities will be branched on before variables without priorities. Variables with higher priorities will be branched on before variables with lower priorities (when the variables have fractional values). To switch off this function, make sure the box is unchecked.

Use MIP priority order When this option is checked, will use the priority order (if it exists) for the next mixed integer optimization.

Do not generate

Use decreasing cost

Use increasing bound range

Use increasing cost per coefficient count

Branch direction

The Branch direction option is used to decide which branch, the up branch or the down branch, should be taken first at each node. For some problems, directing the algorithm to always branch up or down can improve performance.

Branch down Branch down (restricted to lower value)

Algorithm select The algorithm selects the branch direction (default).

Branch up Branch up (restricted to higher value)

Change CPLEX MIP Strategy2 Options

You can further change the MIP strategy options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *MIP Strategy2* tab. This will display the MIP strategy options for CPLEX in the dialog box as shown below:

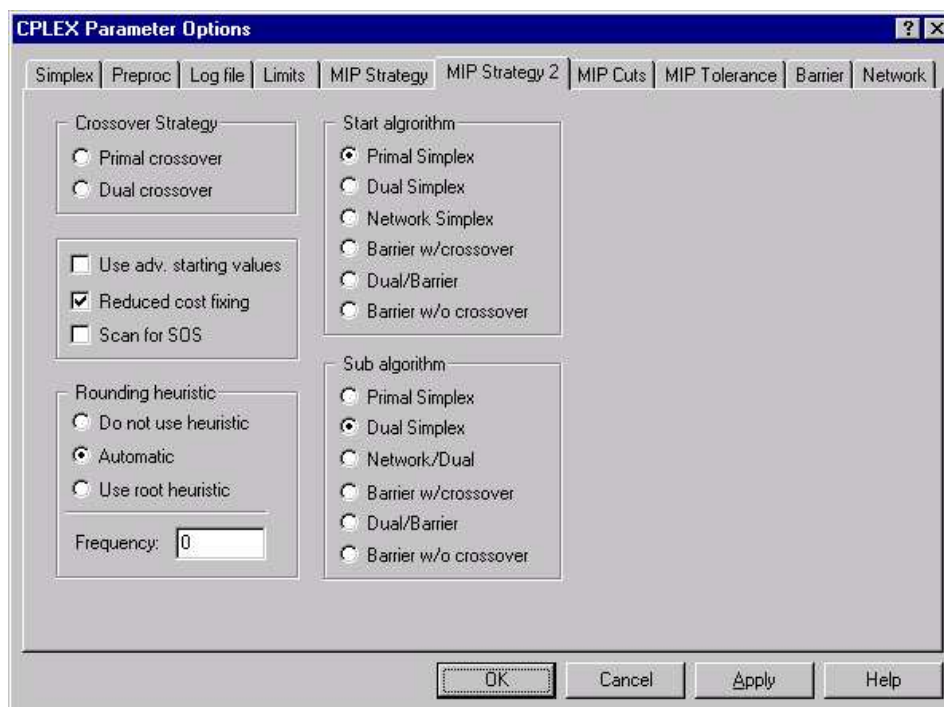


Figure 4.47: The MIP Strategy2 Tab in CPLEX Options Dialog Box

Crossover strategy

Chooses whether to employ Primal or Dual crossover when using Barrier for subproblems.

Use advanced starting values

Used to indicate how the MIP advanced starting values are used at node 0. When set to *On* indicates that the values should be checked to see if they provide an integer feasible solution before starting optimization.

Reduced cost fixing

This option determines whether the reduced cost fixing strategy is to be applied. Reduce cost fixing sets integer values to bounds by considering reduced cost information at sub-problems. This technique can significantly improve MIP performance, but can increase memory usage. By default, reduce cost fixing is *On*, but can be turned off as a memory-saving measure.

Scan for SOS

The Scan for SOS option, when set to *On*, initiates a scan for SOS Type 3 sets and invokes SOS Type 1 branching for these sets of variables. The automatic scan occurs immediately before CPLEX starts optimizing. An SOS Type 3 set results from each equality row with all binaries and +1 or -1 coefficients, and an RHS value of 1 - (number of -1 coefficients). SOS Type 3 sets are a special subset of SOS Type 1 sets which can be identified automatically by CPLEX. The default value for Scan for SOS option is *Off*.

Rounding heuristic

The Rounding heuristic option determines which heuristic should be applied to develop an initial integer solution.

Do not use heuristic Do not use a rounding heuristic.

Automatic The decision on using a heuristic will be automatically determined by looking at the solution to the initial relaxation of MIP.

Use heuristic Rounding heuristic should be used.

Rounding heuristic frequency

The heuristic frequency options determines how often to apply the periodic heuristic. Setting the value to 0, the default, indicates that the periodic heuristic will not be applied at any nodes.

Start algorithm

The Start Algorithm option determines which LP algorithm should be used to solve the initial relaxation of the MIP.

Primal simplex Use primal simplex (default).

Dual simplex Use dual simplex.

Network simplex Use network simplex.

Barrier w/crossover Use barrier with crossover.

Dual/Barrier Dual simplex to iteration limit, then barrier.

Barrier w/o crossover Use barrier without crossover.

Sub algorithm

The sub algorithm option sets the algorithm to be used on subproblems

Primal simplex Use primal simplex.

Dual simplex Use dual simplex (default).

Network/Dual Use network optimizer followed by dual simplex

Barrier w/crossover Use barrier with crossover.

Dual/Barrier Use dual simplex to iteration limit, then barrier.

Barrier w/o crossover Use barrier without crossover.

Change CPLEX MIP Cuts Options

You can change the MIP cuts options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *MIP Cuts* tab. This will display the MIP cuts options for CPLEX in the dialog box as shown below:

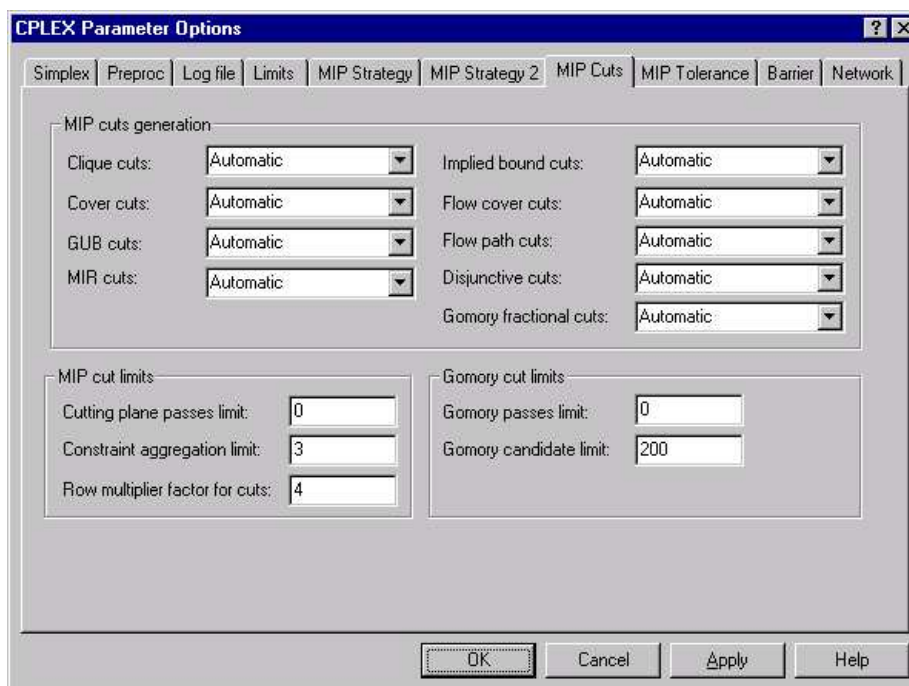


Figure 4.48: The MIP Cuts Tab in CPLEX Options Dialog Box

Clique cuts

Determines whether or not to generate clique cuts for the problem and how aggressively.

<i>Do not generate cuts</i>	Do not generate clique cuts.
<i>Automatic</i>	Generate clique cuts only if it seems to be helping (default)
<i>Moderately</i>	Generate clique cuts moderately.
<i>Aggressively</i>	Generate clique cuts aggressively.

Cover cuts

Determines whether or not to generate cover cuts for the problem and how aggressively.

<i>Do not generate cuts</i>	Do not generate cover cuts.
<i>Automatic</i>	Generate cover cuts only if it seems to be helping (default).
<i>Moderately</i>	Generate cover cuts moderately.
<i>Aggressively</i>	Generate cover cuts aggressively.

Part II Using the MPL Modeling System

GUB cuts

Determines whether or not to generate GUB cuts for the problem and how aggressively.

<i>Do not generate cuts</i>	Do not generate GUB cuts.
<i>Automatic</i>	Generate GUB cuts only if it seems to be helping (default).
<i>Moderately</i>	Generate GUB cuts moderately.
<i>Aggressively</i>	Generate GUB cuts aggressively.

MIR cuts

Determines whether or not mixed integer rounding (MIR) cuts should be generated for the problem and how aggressively.

<i>Do not generate cuts</i>	Do not generate MIR cuts.
<i>Automatic</i>	Generate MIR cuts only if it seems to be helping (default).
<i>Moderately</i>	Generate MIR cuts moderately.
<i>Aggressively</i>	Generate MIR cuts aggressively.

Implied bound cuts

Determines whether or not to generate implied bound cuts for the problem and how aggressively.

<i>Do not generate cuts</i>	Do not generate bound cuts.
<i>Automatic</i>	Generate implied bound cuts only if it seems to be helping (default).
<i>Moderately</i>	Generate implied bound cuts moderately.
<i>Aggressively</i>	Generate implied bound cuts aggressively.

Flow path cuts

Determines whether or not to generate flow path cuts for the problem and how aggressively.

<i>Do not generate cuts</i>	Do not generate flow path cuts.
<i>Automatic</i>	Generate flow path cuts only if it seems to be helping (default).
<i>Moderately</i>	Generate flow path cuts moderately.
<i>Aggressively</i>	Generate flow path cuts aggressively.

Disjunctive cuts

Determines whether or not to generate disjunctive cuts for the problem and how aggressively.

<i>Do not generate cuts</i>	Do not generate disjunctive cuts.
<i>Automatic</i>	Generate disjunctive cuts only if it seems to be helping (default).
<i>Moderately</i>	Generate disjunctive cuts moderately.
<i>Aggressively</i>	Generate disjunctive cuts aggressively.
<i>Very Aggressively</i>	Generate disjunctive cuts very aggressively.

Gomory fractional cuts

Determines whether or not to generate Gomory fractional cuts for the problem and how aggressively.

<i>Do not generate cuts</i>	Do not generate Gomory fractional cuts.
<i>Automatic</i>	Generate Gomory fractional cuts only if it seems to be helping (default).
<i>Moderate</i>	For models requiring some fine tuning of performance.
<i>Aggressive</i>	For models requiring some fine tuning of performance.

Cutting plane passes limit

Sets the upper limit on the number of passes CPLEX performs when generating cutting planes on a MIP model. Positive values give number of passes to perform. And zero means *Automatic* (default).

Constraint aggregation limit

Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts. For most purposes the default of 3 will be satisfactory.

Row multiplier factor for cuts

Limits the number of cuts that can be added. The number of rows in the problem with cuts added is limited to this factor value times the original number of rows. If the problem is presolved, the original number of rows is that from the presolved problem. A factor of 1.0 or less means that no cuts will be generated. Because cuts can be added and removed during the course of optimization, factor may not correspond directly to the number of cuts seen during the node log or in the summary table at the end of optimization.

Gomory passes limit

Limits the number of passes for generating Gomory fractional cuts. At the default setting of 0, CPLEX decides. The parameter is ignored if the Gomory fractional cut parameter is set to a nonzero value.

Gomory candidate limit

Limits the number of candidate variables for generating Gomory fractional cuts. The default is 200.

Change CPLEX MIP Tolerance Options

You can change the MIP tolerance options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *MIP Tolerance* tab. This will display the MIP tolerance options for CPLEX in the dialog box as shown below:

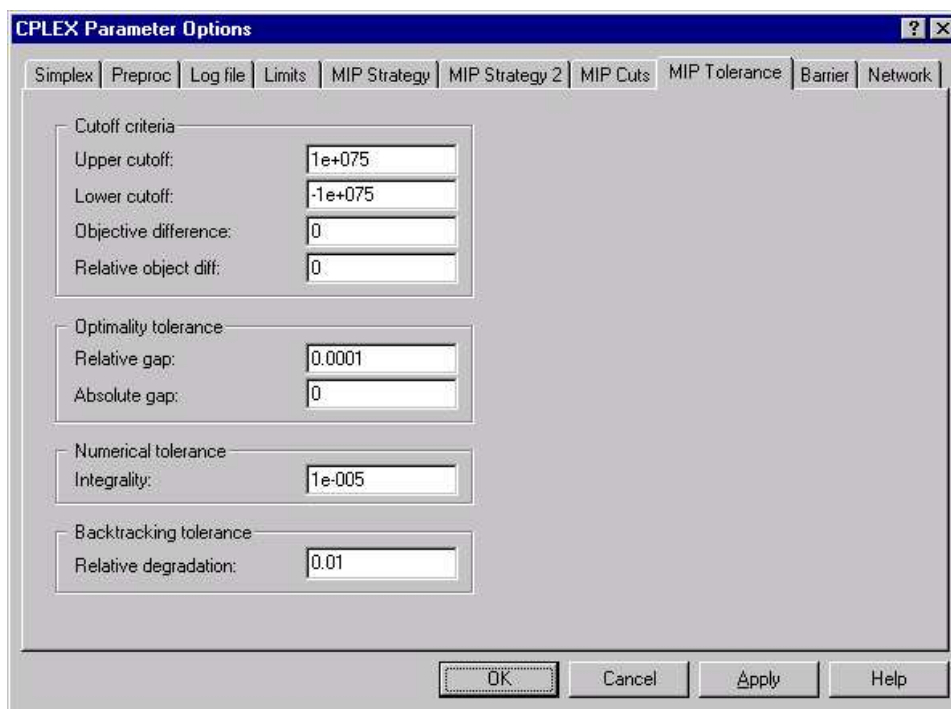


Figure 4.49: The MIP Tolerance Tab in CPLEX Options Dialog Box

Upper cutoff

Used to cut off any nodes that have an objective value above the upper cutoff value, when the problem is a minimization problem. On a continued mixed integer optimization, the smaller of this value and the updated cutoff found during optimization will be used during the next mixed integer optimization. A too-restrictive value for the upper cutoff parameter may result in no integer solutions being found.

Lower cutoff

Used to cut off any nodes that have an objective value below the lower cutoff value, when the problem is a maximization problem. On a continued mixed integer optimization, the larger of this value and the updated cutoff found during optimization will be used during the next mixed integer optimization. A too-restrictive value for the lower cutoff option may result in no integer solutions being found.

Objective difference

The objective difference value is used to update the cutoff each time a mixed integer solution is found. This absolute value will be subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the mixed integer optimization to ignore integer solutions that are not at least this amount better than the one found so far. The objective difference value can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this option at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed. Negative values for this option will result in some integer solutions that are worse than or the same as those previously generated, but will not necessarily result in the generation of all possible integer solutions.

Relative object diff

The relative objective difference value is used to update the cutoff each time a mixed integer solution is found. The value is multiplied by the absolute value of the integer objective and subtracted from (added to) the newly found integer objective when minimizing (maximizing). This forces the mixed integer optimization to ignore integer solutions that are not at least this amount better than the one found so far. The relative objective difference option can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this parameter at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed. If both relative objective difference and objective difference are nonzero, the value of the objective difference will be used.

Relative gap

The relative tolerance on the gap between the best integer objective and the objective of the best node remaining.

Absolute gap

The absolute tolerance on the gap between the best integer objective and the objective of the best node remaining.

Integrality tolerance

The integrality tolerance specifies the amount by which an integer variable can be different than an integer and still be considered feasible.

Backtrack factor

The backtracking factor controls how often backtracking is done during the branching process. At each node, CPLEX compares the objective function value or estimated integer objective value to these values at parent nodes; the backtracking parameter dictates how much relative degradation is tolerated before backtracking. Allowable values are any positive number. Low values tend to increase the amount of backtracking, which makes the search process more of a pure best-bound search. Higher values, greater than 1.0, tend to decrease backtracking, making the search more of a pure depth-first search. While the default value is good for many models, increased backtracking (lower values) often pays off on models that have expensive sub-problems.

Change CPLEX Barrier Options

You can change the barrier options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *Barrier* tab. This will display the barrier options for CPLEX in the dialog box as shown below:

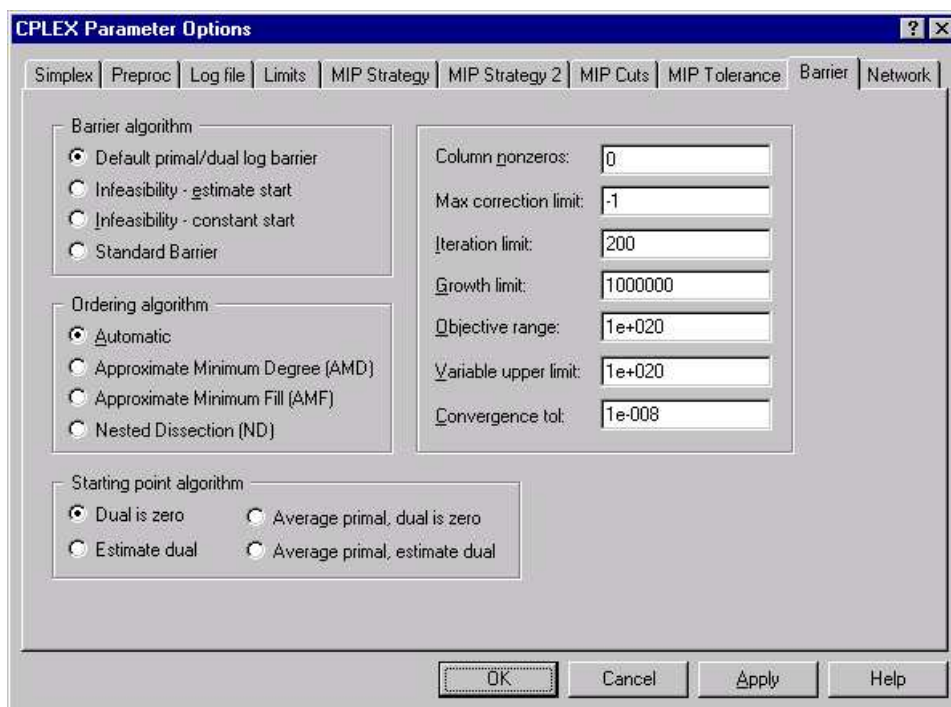


Figure 4.50: The Barrier Tab in CPLEX Options Dialog Box

Barrier algorithm

The default barrier algorithm is almost always fastest. However, on problems that are primal or dual infeasible, the default algorithm may not work as well as alternatives. The two alternative algorithms may eliminate numerical difficulties related to infeasibility, but will generally be slower

<i>Primal Dual Log</i>	Use standard algorithm. This is the default
<i>Infeasibility-Estimate</i>	Use infeasibility-estimate start
<i>Infeasibility-Constant</i>	Use infeasibility-constant start
<i>Standard Barrier</i>	Use for problems other than MIP, it is fastest.

Ordering algorithm

The Ordering option sets the algorithm to be used to permute the rows of the constraint matrix in order to reduce fill in the Cholesky factor.

<i>Automatic</i>	This option attempts to choose the most effective of the available ordering methods. It usually requires more ordering runtime than any of the alternatives, but it typically chooses the best ordering. The ordering runtime is usually small relative to the total solution time, and a better ordering can lead to smaller total solution time.
<i>AMD</i>	The AMD algorithm provides good quality orderings in moderate ordering runtime.
<i>AMF</i>	The AMF algorithm usually gives better orderings than AMD (usually 5-10% smaller factors), but requires somewhat more runtime (10-20%).
<i>ND</i>	The ND algorithm produces significantly better orderings than AMD or AMF. Ten-fold reductions in barrier solvers runtimes have been observed for some problems. This option sometimes produces worse orderings, though, and it requires much more ordering runtime.

Starting point algorithm

This option sets the algorithm to be used to compute the initial starting point for the barrier solver. The default starting point is satisfactory for most problems. Other starting points may provide better performance for certain problems, or provide better convergence properties for the barrier algorithm. Since the default starting point is tuned for primal problems, using the other starting points may be worthwhile in conjunction with the Presolve Pass Dual Problem option.

<i>Dual is zero</i>	Default primal, dual pi is 0
<i>Estimate Dual</i>	Default primal, estimate dual
<i>Avg. primal, Dual is zero</i>	Primal average, dual pi is 0
<i>Avg. primal, Estimate dual</i>	Primal average, estimate dual

Column nonzeros

The Column Nonzeros value is used in the recognition of dense columns. If columns (in the presolved and aggregated problem) exist with more entries than the value of this option, these columns will be considered "dense" and CPLEX Barrier will treat the dense columns specially in order to reduce their effect. At the default setting of 0 this option is determined automatically, considering factors such as the size of the problem. If a number greater than 0 is entered, this number will be used as the "cutoff" number of entries for considering columns to be dense.

Note: If the problem (after Presolve and Aggregator) contains less than 400 rows, dense column handling will not be initiated, regardless of the column nonzero setting.

Max correction limit

This option sets the maximum number of centering correction done on each iteration. By default the barrier solver automatically computes an estimate value for this parameter (the computed value can be observed by setting the barrier progress info log option to diagnostics information in the log file option dialog box). When the using the default barrier algorithm, if the computed value is zero setting the value to an explicit value greater than zero may improve the numerical performance of the algorithm at the expense of computation time.

- 1 Automatically determined (default)
- 0 None
- >0 Maximum number of corrections

Iteration limit

The iteration limit sets the number of barrier iterations before termination. When set to 0, no barrier iterations will occur, but problem "setup" will occur and information about the setup will be displayed (Cholesky factorization information).

Growth limit

The growth limit is used to detect unbounded optimal faces. At higher values, the barrier algorithm will be less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical difficulties if the problem has an unbounded face. The default value is 1e-6.

Objective range

The objective range value sets the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems. The default value is 1e+15.

Variable upper

The variable upper value sets the upper bound for all variables that have infinite upper bounds. This limit is used to prevent difficulties associated with unbounded optimal faces. The default value is 1e+20.

Convergence tol

The convergence tolerance sets the tolerance on complementarity for convergence. The barrier algorithm will terminate with an optimal solution if the relative complementarity is smaller than this value. The default value is 1e-8.

Change CPLEX Network Options

You can change the network options for CPLEX by choosing *CPLEX parameters* from the *Options* menu and then pressing the *Network* tab. This will display the network options for CPLEX in the dialog box as shown below.

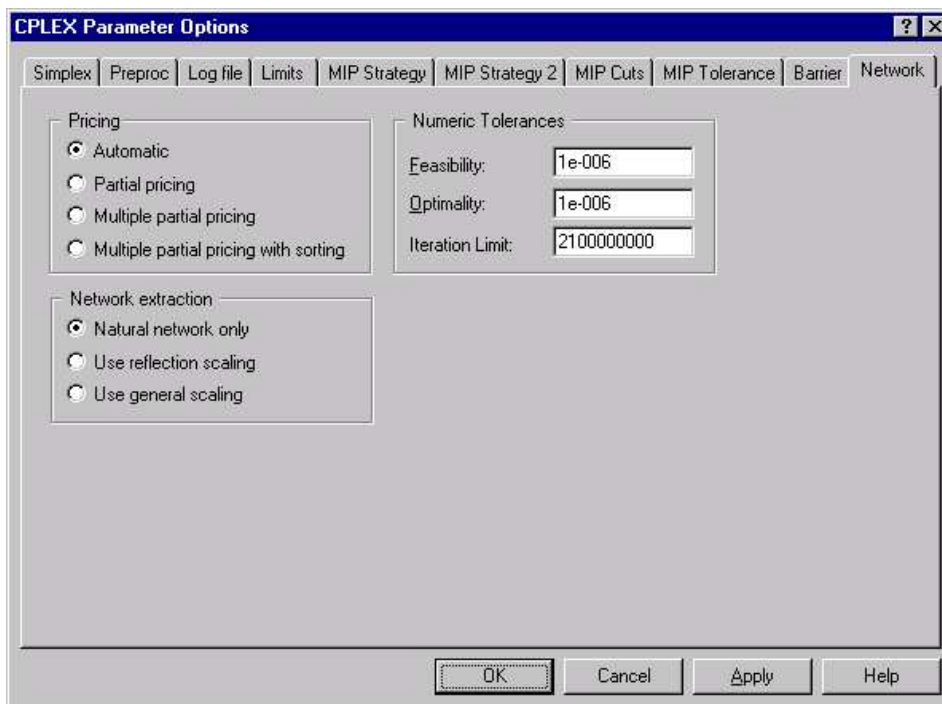


Figure 4.51: The Network Tab in CPLEX Options Dialog Box

Pricing

The default (Automatic) shows best performance for most problems, and currently is equivalent to *Multiple partial pricing with sorting*.

Automatic (default)

Partial pricing

Multiple partial pricing

Multiple partial pricing with sorting

Part II Using the MPL Modeling System

Network extraction

The Network extraction option selects the level of network extraction for network simplex optimizations.

- Natural network only* Only natural networks are extracted (default).
- Use reflection scaling* Larger networks are extracted using reflection scaling.
- Use general scaling* Larger networks are extracted using general scaling.

Numeric tolerances

- Feasibility* The feasibility tolerance specifies the degree to which a problem's flow value may violate its bounds. This tolerance influences the selection of an optimal basis and can be reset to a lower value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, CPLEX may falsely conclude that a problem is infeasible. If you encounter reports of infeasibility during Phase II of the optimization, a small adjustment in the feasibility tolerance may improve performance. The value can be any number between 0.0001 and $1e-11$.
- Optimality* The optimality tolerance specifies the amount a reduced cost may violate the criterion for an optimal solution. The value can be any number between 0.0001 and $1e-11$.
- Iteration Limit* Sets the maximum number of iterations to be performed before the algorithm terminates without reaching optimality. The value can be any non-negative integer.

Change XPRESS Simplex Options

You can change the simplex options for XPRESS by choosing *XPRESS parameters* from the *Options* menu and then pressing the *Simplex* tab. This will display the simplex options for XPRESS in the dialog box as shown below:

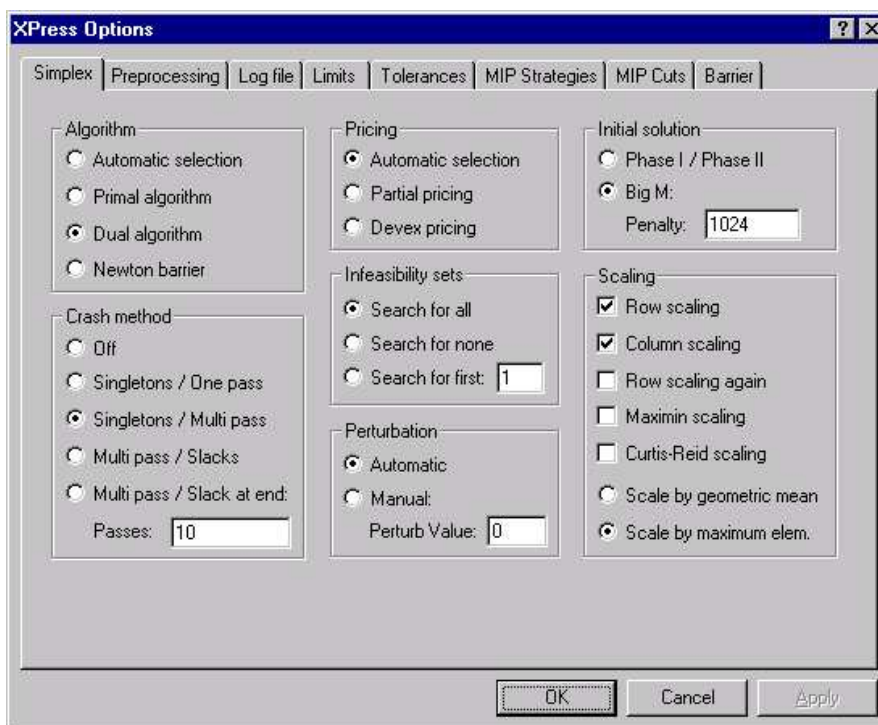


Figure 4.52: The Simplex Tab in XPRESS Options Dialog Box

Algorithm

The algorithm to be used by the optimizer to solve the problem can be specified here. The default value is *Automatic selection*. Following are the four possible selections:

<i>Automatic selection</i>	Dual simplex for LPs and IPs; Newton barrier for QP.
<i>Primal algorithm</i>	Use primal simplex algorithm.
<i>Dual algorithm</i>	Use dual simplex algorithm.
<i>Newton barrier</i>	Use Newton barrier algorithm

Part II Using the MPL Modeling System

Crash method

The type and severity of the crash to be performed for primal simplex can be choosing one of the following options:

<i>Off</i>	Turns off all crash procedures.
<i>Singletons / One pass</i>	For singletons only, one pass.
<i>Singletons / Multi pass</i>	For singletons only, multi pass (default).
<i>Multi pass / Slacks</i>	
<i>Multi pass / Slack at end</i>	Multiple passes through the matrix considering slacks. You can specify how many passes to perform.

Pricing

The *Pricing* option for primal simplex has the following possible selections:

<i>Automatic selection</i>	The pricing strategy is to be decided automatically (default).
<i>Partial pricing</i>	Partial pricing is to be used.
<i>Devex pricing</i>	Devex pricing is to be used.

Infeasibility sets

Infeasibility sets controls the number of irreducible infeasible sets to be found. It has the following possible selections:

- Search for all* (default).
- Search for none*.
- Search for first n*.

Perturbation

The *Use perturbation* is used to select whether to use perturbation or not, prior to optimization. You can choose either *Automatic* perturbation or *Manual* perturbation; if you choose *Manual*, you can set the perturbation value (default is 0, i.e., no perturbation).

Initial solution

The *Initial Solution* allows you to choose between the *Phase I/Phase II* method or the *Big M* method (the default) to obtain an initial feasible solution.

<i>Phase I / Phase II</i>	For PhaseI/PhaseII.
<i>Big M</i>	Big M method to be used. You can set the infeasibility <i>Penalty</i> which has the default of 1024 (default).

Scaling

It is always worth striving to create a well-scaled matrix during the formulation stage. This is not always easy to do, and so automatic scaling can be applied to improve the numerical stability, if the range of coefficient values of a matrix is very large. The following are the five available scaling techniques:

Row scaling.

Column scaling.

Row scaling again.

Maximin scaling.

Curtis-Reid scaling.

Furthermore, you can choose between scaling by *geometric mean* or by *maximum element*. The default is *Row and Column scaling, by maximum element*.

Scaling of integer entities is not supported, though *XPRESS* will scale the continuous variables in a MIP problem. This means that you should be careful about the scale of the integer entities when formulating MIP problems.

Change XPRESS Preprocessing Options

You can change the preprocessing options for XPRESS by choosing *XPRESS parameters* from the *Options* menu and then pressing the *Preprocessing* tab. This will display the preprocessing options for XPRESS in the dialog box as shown below:

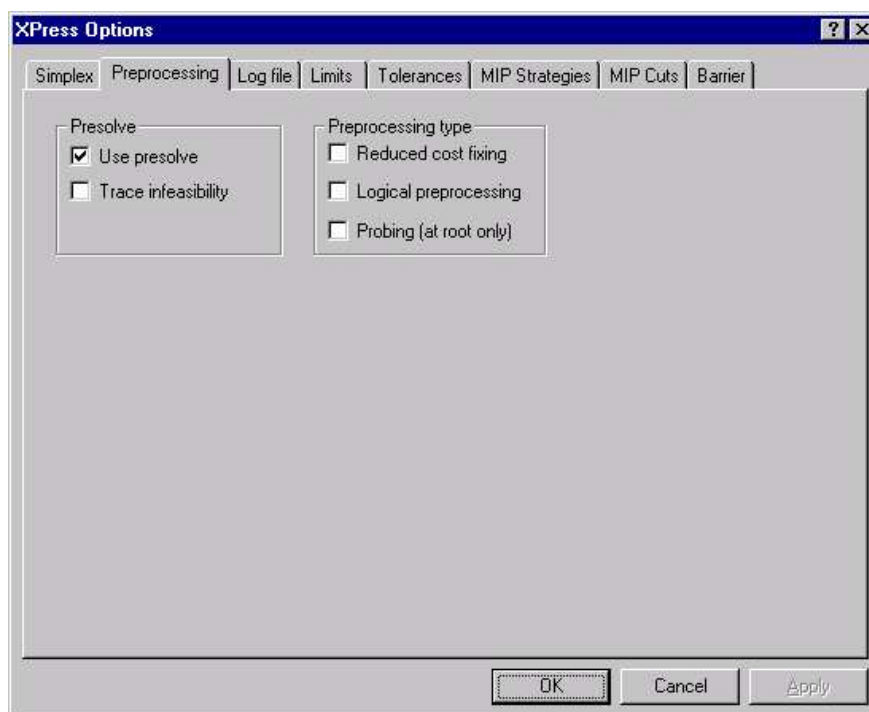


Figure 4.53: The Preprocessing Tab in XPRESS Options Dialog Box

Use presolve

The presolve facility can greatly improve performance by modifying the user's matrix so that it is easier to solve. The presolve algorithms identify and remove redundant rows and columns, thus reducing the size of the matrix. If the model contains global entities, integer presolve methods such as bound fixing and coefficient tightening are also applied to tighten the LP relaxation.

When a solution to a presolved problem has been found it is internally *postsolved* so that the user can access a solution to the original problem.

If *Use presolve* is *On*, the optimizer will begin by preprocessing the matrix. This reduces the active matrix size by identifying redundant rows and columns by inspection and modifying the matrix in order to make it easier to optimize.

Trace infeasibility

Control of the infeasibility diagnosis during presolve. If *On*, the logical deductions made by presolve to deduce infeasibility will be displayed.

Reduced cost fixing

Reduced cost fixing at each node. Default is *On*.

Logical preprocessing

Logical preprocessing at each node. Default is *On*.

Probing (at root only)

Probing set at the top node. Default is *On*.

If any of these three above options are selected, integer preprocessing for MIP problems will be performed at each node of the branch-and-bound search tree (including the top node). If a variable is fixed at a node, it remains fixed at all its child nodes but it is not deleted from the matrix (unlike the variables fixed by presolve).

Change XPRESS Log File Options

You can change the log file options for XPRESS by choosing *XPRESS parameters* from the *Options* menu and then pressing the *Log file* tab. This will display the log file options for XPRESS in the dialog box as shown below:

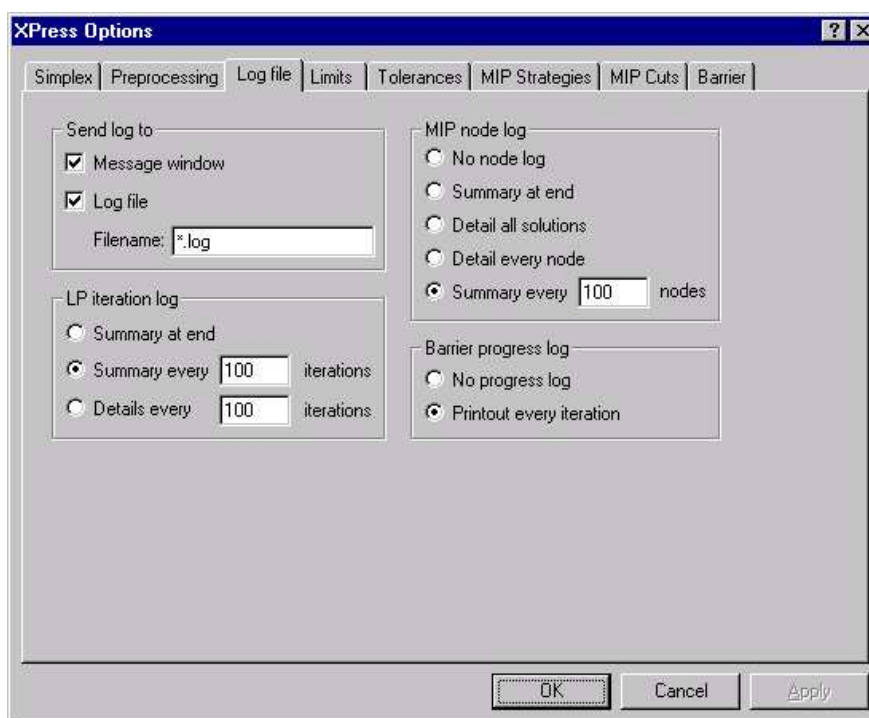


Figure 4.54: The Log File Tab in XPRESS Options Dialog Box

Send log to:

Message window: While the solver is optimizing, the iteration log information will be sent to the *Message Window*.

Log file: While the solver is optimizing, the iteration log information will be sent to a log file.

Log filename: Specifies the filename the solver will use for the log file. If the filename given contains asterisks '*' instead of the name, like the default entry '*.log', the solver will use the name of model file with the extension given.

LP iteration log

The *LP iteration log* controls the amount of information that is sent to either the log window and/or the log file. The following are the possible selections:

Summary at end

Summary every n iterations.

Details every n iterations.

The default is summary at every 100 iterations.

MIP node log

The *MIP Node log* option controls the frequency of the node logging. This option is useful for monitoring the progress of a problem that requires many nodes to solve. The following are the possible selections:

No node log.

Summary at end.

Detail all solutions.

Detail every node.

Summary every n nodes.

The default is summary at every 100 nodes.

Barrier progress info log

The *Barrier Progress Info Log* option has the following possible selections:

No progress log.

Printout every iteration (default).

Change XPRESS Limit Options

You can change the limit options for XPRESS by choosing *XPRESS parameters* from the *Options* menu and then pressing the *Limits* tab. This will display the limit options for XPRESS in the dialog box as shown below:

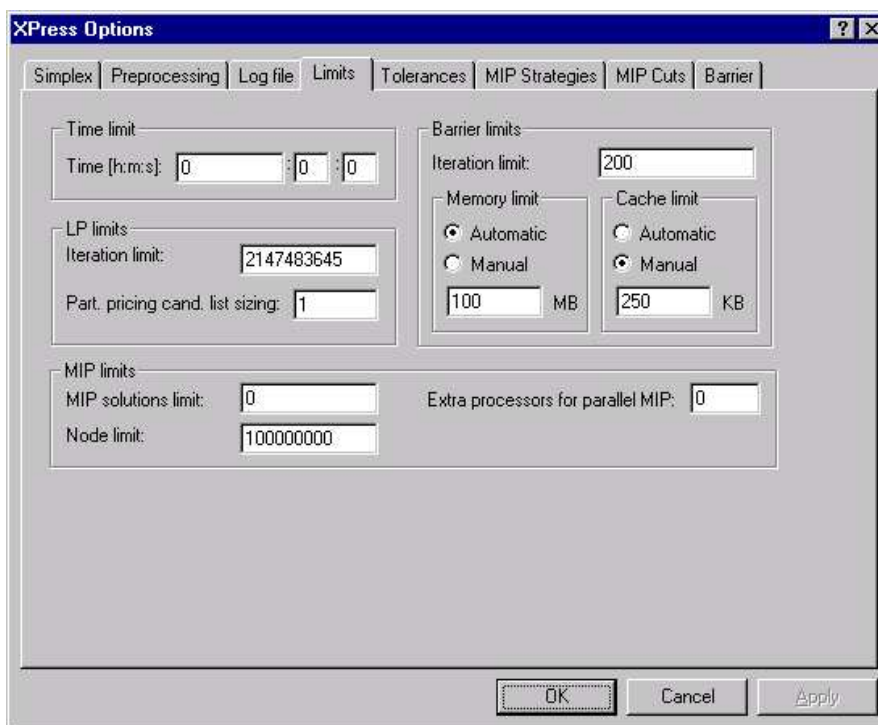


Figure 4.55: The Limits Tab in XPRESS Options Dialog Box

Time limit

The *time limits* option allows you to specify the maximum time for the optimization.

LP iteration limit

The *LP iteration limit* option specifies iteration limit for the simplex algorithm, summed up over all nodes. The default value is 2147483654.

Part. pricing cand. list sizing

Partial pricing candidate list sizing parameter. The default value is 1.0.

Barrier iteration limit

The *barrier iteration limit* option specifies the maximum number of Newton Barrier iterations. The default value is 200.

Barrier memory limit

Barrier memory limit specifies the amount of memory in megabytes to be used by the barrier algorithm. The default of *Automatic* indicates the memory should be determined automatically. The *Manual* option allows you to enter the memory amount in *MB*.

Barrier cache limit

Barrier cache limit specifies the amount of memory in kilobytes to be used by the barrier algorithm. The default setting *Automatic* indicates the memory should be determined automatically, if possible (if not, a value of 512 kB is assumed). The *Manual* option allows you to enter the memory amount in *KB*.

MIP solutions limit

Maximum number of integer solutions to find. Default is 0, which means no limit.

Node limit

Maximum number of nodes in Branch and Bound search. Default is 100000000.

Extra processors for parallel MIP

Number of slave processors to use for the parallel MIP search. Default is 0.

Change XPRESS Tolerance Options

You can change the tolerances options for XPRESS by choosing *XPRESS parameters* from the *Options* menu and then pressing the *Tolerances* tab. This will display the tolerance options for XPRESS in the dialog box as shown below:

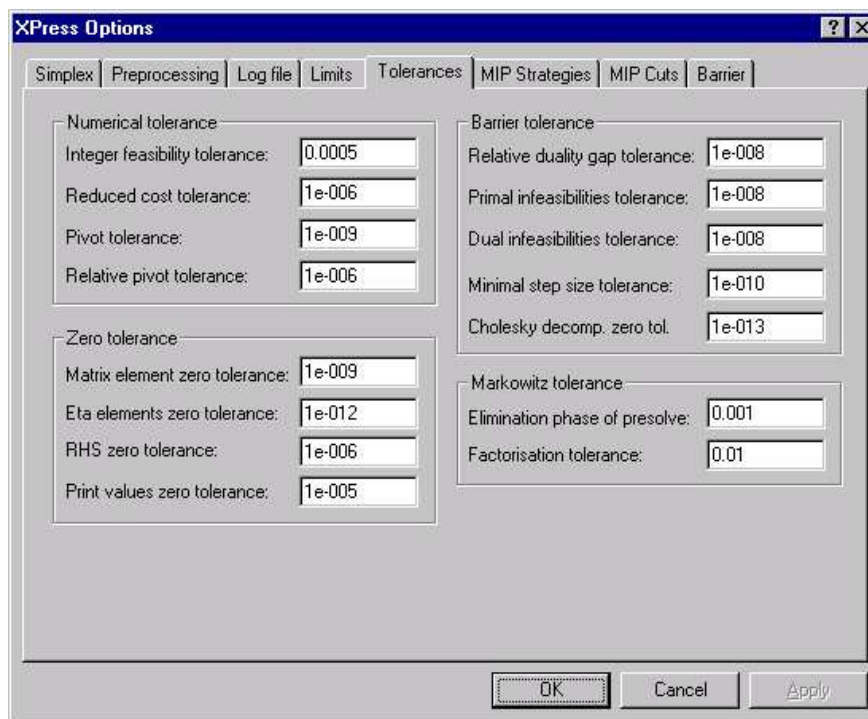


Figure 4.56: The Tolerances tab in XPRESS Options Dialog Box

Numerical tolerance

There are four numerical tolerances that can be set:

Integer feasibility tolerance (default value is $5e-6$);

Reduced cost tolerance (default value is $1e-6$);

Pivot tolerance (default value is $1e-9$) and

Relative pivot tolerance (default value is $1e-6$).

Zero tolerance

There are four zero tolerances that can be set:

Matrix element zero tolerance (default value is 1e-9);

Eta elements zero tolerance (default value is 1e-13);

RHS zero tolerance (default value is 1e-6) and

Print values zero tolerance (default value is 1e-5).

Barrier tolerance

There are five barrier tolerances that can be set:

Relative duality gap (default is 1e-8);

Primal infeasibilities tolerance (default is 1e-8);

Dual infeasibilities tolerance (default is 1e-8);

Minimal step size tolerance (default is 1e-10) and

Cholesky decomp. zero tolerance (default is 1e-15).

Markowitz tolerance

There are two markowitz tolerance options that can be set:

Elimination phase of presolve (default is 0.001) and

Factorization tolerance (default is 0.01).

Change XPRESS MIP Strategy Options

You can change the MIP Strategy options for XPRESS by choosing *XPRESS parameters* from the *Options* menu and then pressing the *MIP Strategies* tab. This will display the MIP strategy options for XPRESS in the dialog box as shown below:

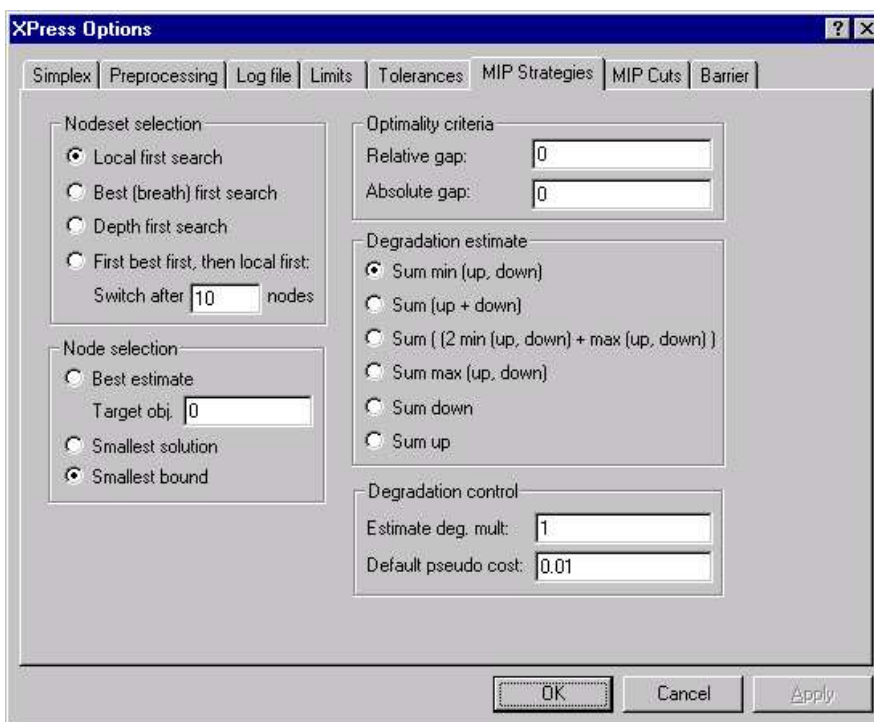


Figure 4.57: The MIP Strategies Tab in XPRESS Options Dialog Box

Nodeset selection

The *nodeset selection* option has the following possible selections:

- | | |
|--|---|
| <i>Local first search</i> | Choose among the two descendant nodes, if none among all active nodes (default). |
| <i>Best first search</i> | All nodes are always considered. |
| <i>Depth first search</i> | Depth-first search exploring both descendants first. |
| <i>First best first, then local first,</i> | |
| <i>Switch after n nodes</i> | All nodes are considered for the first <i>n</i> nodes, after which local first search is resumed. |

Node selection

The *node selection* option has the following possible selections:

<i>Best estimate</i>	If a target object function has not been set, choose the node with the best estimate target object function for global. If a target objective function has is been set (by the user or from a previous IP solution), the choice is based on the Forrest-Hirst-Tomlin Criterion. The <i>Target obj.</i> function used in “best estimate” node selection technique. This is set automatically after solving the LP relaxation unless set by the user.
<i>Smallest solution</i>	Always choose the node with smallest estimated solution.
<i>Smallest bound</i>	Always choose the node with smallest bound (default).

Relative gap

The relative MIP optimality stopping criterion. The MIP search will stop if the relative optimality gap, $ABS(\text{best solution} - \text{best bound}) / \text{best bound}$, is less than or equal to this criterion value. The default is 0.0.

Absolute gap

The absolute MIP optimality stopping criterion. The MIP search will stop if the absolute optimality gap, $ABS(\text{best solution} - \text{best bound})$, is less than or equal to this criterion value. The default is 0.0.

Degradation estimate

Degradation estimate is the node selection degradedator estimate control. The *Degradation estimate* option has the following possible selections:

- Sum min (up, down)* (default)
- Sum (up + down)*
- Sum ((2 min (up,down) + max (up, down))*
- Sum max (up, down)*
- Sum down*
- Sum up*

Estimate deg. mult

Factor to multiply estimated degradations by, default is 1.0.

Default pseudo cost

Default pseudo cost used in node degradation estimation, default is 0.01.

Change XPRESS MIP Cuts Options

You can change the MIP cuts options for XPRESS by choosing *XPRESS parameters* from the *Options* menu and then pressing the *MIP Cuts* tab. This will display the MIP cuts options for XPRESS in the dialog box as shown below:

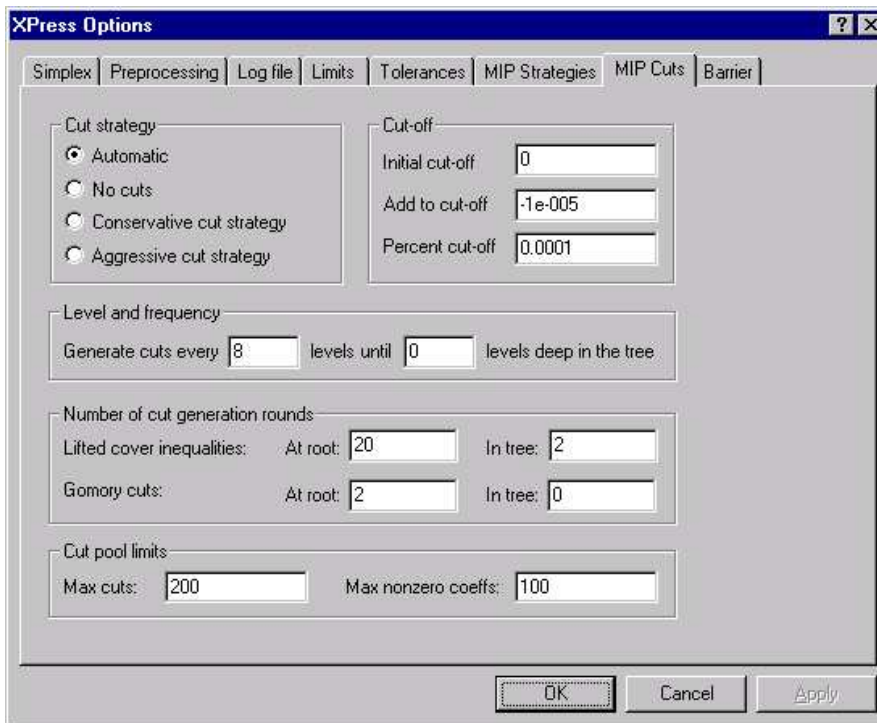


Figure 4.58: The MIP Cuts Tab in XPRESS Options Dialog Box

Cut strategy

XPRESS automatically generates and applies cuts tailored to your problem. Adding cuts during the global search may lead to bound changes on variables, in addition to those imposed by the branch-and-bound algorithm, which come in to effect when branching.

The *Cut strategy* option has the following possible selections: *Automatic* (default)/*No cuts*/*Conservative cut strategy*/*Aggressive cut strategy*.

Cut-off

Initial cut-off: Nodes in the IP tree search with an LP objective value worse than *Cut-off* value are fathomed, i.e., not explored any further. Set it to the value of a known IP feasible solution (although this will prevent *XPRESS* finding that solution) or any objective value you must better in the IP search.

Add to cut-off: After an IP solution is found, Cut-off value is updated to be the value of the IP solution plus Add to cut-off value. So *XPRESS* will not bother looking for other IP solutions within Add to cut-off value of the solution it has found. Add to cut-off value would normally be positive for maximization solutions; and negative for minimization solutions.

Percent cut-off: After an IP solution is found, *Cut-off* value is updated to be the value of the IP solution plus *Percent cut-off* value. So *XPRESS* will not bother looking for other IP solutions within *Percent cut-off* value of the solution it has found.

Level and frequency

The *level and frequency* option has the following possible selections. *Generate cuts every n levels:* specifies the frequency at which cuts are generated in the tree search. If the depth of the node modulo this value is zero, then cuts will be generated. The default is 8. *Generate cuts until n levels deep in the tree:* specifies the maximum depth in the tree search at which cuts will be generated. The default is 0, so cuts are not generated in the tree by default.

Lifted cover inequalities

LCI at root Number of rounds of lifted cover inequalities at the top node, default is 2.

LCI in tree Number of rounds of lifted cover inequalities at nodes in the tree, default is 2.

Gomory cuts

Gomory at root Number of rounds of Gomory cuts at the top node, default is 2.

Gomory in tree Number of rounds of Gomory cuts at nodes in the tree, default is 2.

Max cuts

The *max cuts* option contains the maximum number of cuts in the cut pool. For maximum efficiency, the space-allocating *max cuts* option should be specified by the user if their values are known. If this is not done, resizing will occur automatically, but more space may be allocated than the user requires. The default value is 200 cuts.

Max nonzero coeffs.

The *max nonzero coeffs.* option contains the maximum number of nonzero coefficients in the cut pool.

Change Barrier Options for XPRESS

You can change the barrier options for XPRESS by choosing *XPRESS parameters* from the *Options* menu and then pressing the *Barrier* tab. This will display the barrier options for XPRESS in the dialog box as shown below:

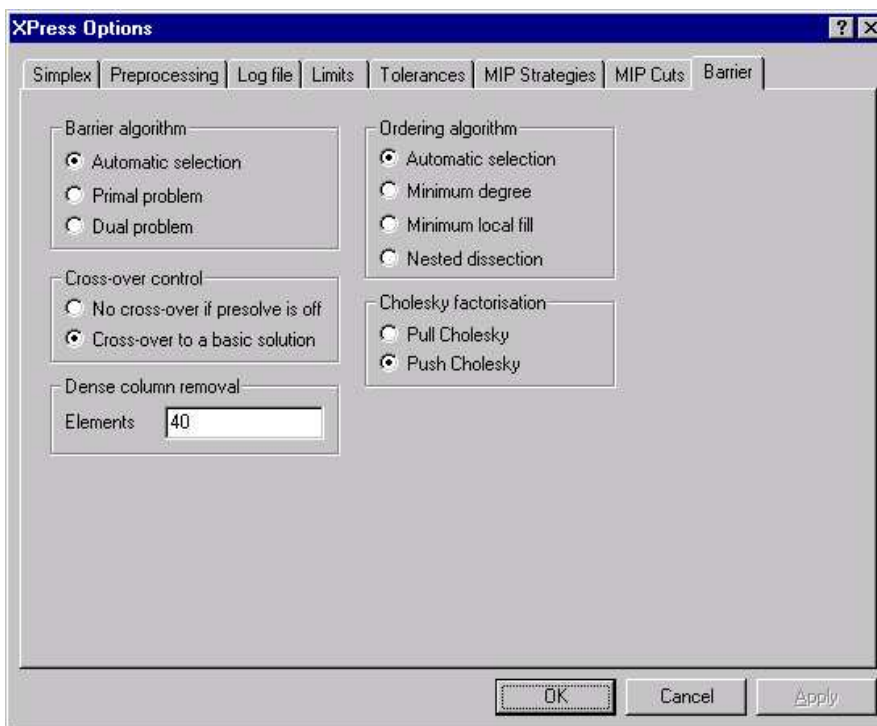


Figure 4.59: The Barrier Tab in XPRESS Options Dialog Box

Barrier algorithm

Selects whether the algorithm solves the primal or dual problem. The *barrier algorithm* option has the following possible selections:

Automatic selection (default).

Primal problem.

Dual problem.

Cross-over control

The *cross-over control* option has the following possible selections:

No cross-over (only available if presolve is *Off*).

Cross-over to a basic solution (default).

Dense column removal

Columns with more elements than this value are considered to be dense, and a special procedure is used to handle them in the *Cholesky factorization*.

Ordering algorithm

Specifies the ordering algorithm for the *Cholesky factorization*. The *ordering algorithm* option has the following possible selections:

Automatic selection (default).

Minimum degree.

Minimum local fill.

Nested dissection.

Cholesky factorization

Specifies the type of *Cholesky factorization* used. The available options are:

Pull Cholesky (default).

Push Cholesky.

Change Solver Parameter Options

You can change various native solver parameter options by choosing *Solver Parameters* from the *Options* menu. You will be presented with a submenu where you can select which option dialog box you want to display. Depending on how the solver handles options, there are two types of dialog boxes available. If **MPL** can, as with CPLEX, control the options directly through memory, you will see a separate menu containing all the applicable dialog boxes. If **MPL**, on the other hand, controls the options through a text file, such as with XA and FortMP, you will see a dialog box called *<Solvername> Option Parameters* as shown here in Figure 4.61.

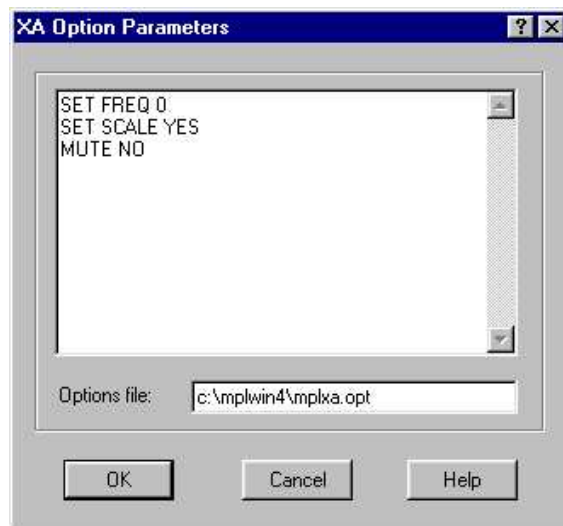


Figure 4.60: XA Option Parameters Dialog Box

This dialog box works very similarly to a text editor, where you can insert new options and change or remove existing options. When you are finished, press the *OK* button to let **MPL** save the options to the filename listed in *Options file* input box. If you do not want to save the options press *Cancel*.

Solver Options List Dialog Box

The Solver Options List dialog box allows you to change various options for supported solvers in a text editor window.

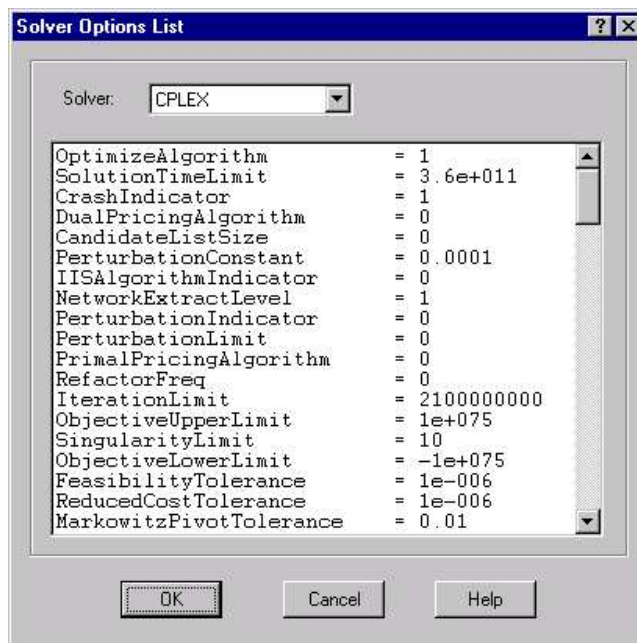


Figure 4.61: Solve Option List Dialog Box

The *Solver* drop-down list allows you to choose which solver you are changing the parameter options for. When you select the solver, **MPL** automatically pulls in all the different options for that particular solver and lists them in the text edit box below. There you can change any of the options that you may need. Please refer to the documentation for each solver or contact Maximal Software for further information on each option.

Setup Solvers for the Run Menu

You can change which solvers are shown in the *Run* menu by choosing *Solver Menu* from the *Options* menu. This will display the *Solver Menu Setup* dialog box shown below in Figure 4.63.

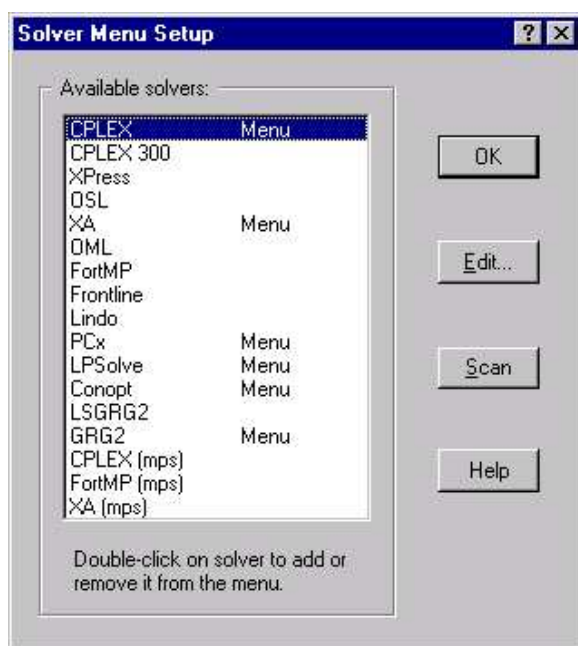


Figure 4.62: Solver Menu Setup Dialog Box

The list box shows all solvers that are supported by the current version of **MPL**. Depending on which solvers you have installed on your machine, double-click on a solver to either add or remove it from the menu. Those solvers that are currently in the menu have the word *Menu* listed in the second column. The solvers that have *(mps)* directly after the name are DOS legacy solvers.

If you need to change some of the setup options for a solver, select the solver in the list box and then press the *Edit* button. This will display the *Solver Setup Options* dialog box which is explained extensively in the section on the next page.

If you want **MPL** to search your hard disk for supported solvers, you can do so by pressing the *Scan* button. This option can be especially useful when you are not sure where on the hard disk solvers have been installed and you want **MPL** to locate them automatically and set them up.

Change Setup Options for Solvers

If you need to change some of the setup options for a solver, select the solver first in the *Solver Menu Setup* dialog box described on the previous page and then press the *Edit* button. This will display the *Solver Setup Options* dialog box with the options for that solver. These setup options have been set to the correct defaults and under most circumstances do not need to be changed.



Figure 4.63: The Solver Setup Options Dialog Box

The following is a list of the setup options you can change.

Solver ID: Selects the solver whose entry is being changed. This entry is almost never changed after it has been set to a certain solver.

Menu Name: Sets the menu name as it appears on the *Run | Solve* menu. The default is the name of the solver. When the solver is a DOS solver that uses MPS input, the default menu name is the name of the solver followed by (*mps*).

DLL Filename: Specifies the name and location of DLL solvers. **MPL** will automatically fill this field when it locates supported solvers at startup.

Solver type

Specifies whether the solver is a Windows DLL solver (default), DLL driver, solver library built into **MPL**, or a legacy DOS solver. In most cases, the default option DLL is the correct one.

Part II Using the MPL Modeling System

DLL solver options

DLL solvers have two further options that are displayed when solver type is set to DLL or Driver.

License: In some cases solvers need to receive a license string when they are being loaded into memory which can be specified here.

Server: Specifies network location for external server solvers.

DOS solver options

DOS solvers have selected options that are displayed when solver type is set to DOS.

PIF Filename: Sets the PIF filename for a DOS solver. **MPL** comes with a separate PIF file for each DOS solver it supports. You can change options of the PIF file using the *PIF Editor* program that comes with Windows (see the Windows documentation for details). You can also create a new PIF file and give the filename here.

Input Filename: Specifies the filename **MPL** will use for the input file for the solver. If the filename given contains star '*' instead of the name, like *.mps, **MPL** will use the name of the model file with the extension given. If the solver needs to be told which type the input is, it can be specified in the *Type* column.

Output Filename: Specifies the filename **MPL** will use for the output file from the solver. If the filename given contains '*' instead of the name, like *.out **MPL** will use the name of the model file with the extension given. If the solver needs to be told which type the output is, it can be specified in the *Type* column.

Pause after solve: After the solver has solved the problem in a DOS window, this option can be used to let the program wait until you press a key before closing the window. This is especially useful when something goes wrong during the optimization process and you want to see the error message from the solver before continuing.

4.10 The Window Menu

The *Window* menu is used when you need to change the layout of windows that are currently opened or minimized. It also allows you to close or select open windows.

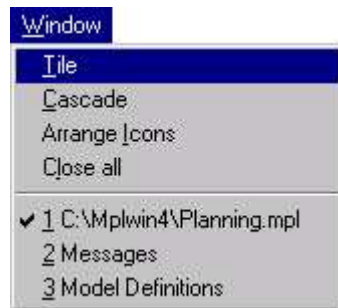


Figure 4.64: The Window Menu

Tile	- Divides the screen equally between windows
Cascade	- Arranges window on top of each other
Arrange Icons	- Arranges minimized windows at the bottom
Close All	- Closes all windows

Tile and Cascade Windows

To divide the main window equally between open windows, choose *Tile* from the *Window* menu. To cascade or stack all open windows on top of each others so that you can see the title of each window choose *Cascade* from the *Window* menu.

Arrange Minimized Icons

To arrange at the bottom all the windows that have been minimized down to an icon, choose *Arrange Icons* from the *Window* menu.

Close All Windows

To close all editor, view, and graph windows that are open, choose *Close all* from the *Window* menu.

List of Open Windows

MPL keeps a list of all open windows in the window menu that can be used to quickly search for and switch between windows.

4.11 The Help Menu

The *Help* menu is used when you need to access the **MPL** Help System. You can also use the shortcut *F1* to access the help system.



Figure 4.65: The Help Menu

Using the MPL Help System

To open the main help system window for **MPL**, select *Topics* from the *Help* menu. This will display the help window for **MPL** where you can select the help topic you wish to view. The help window contains three tabs, *Contents*, *Index*, and *Find*. The *Contents* tab lists all the help topics in a hierarchical tree structure, while the *Index* and the *Find* tabs allow you to search for the help topic you want either by keyword or by words or phrases in the help file.

You can also display the help window with the index tab on top by selecting the *Search for Help On* command directly from the *Help* menu.

The Contents tab

The *Contents* tab of the help window shows all the available topics in the help system in a hierarchical tree structure. Each item shown as a book in the tree corresponds to a category or a section in the help while the items containing the actual help topics are represented by a page with a question mark on it.

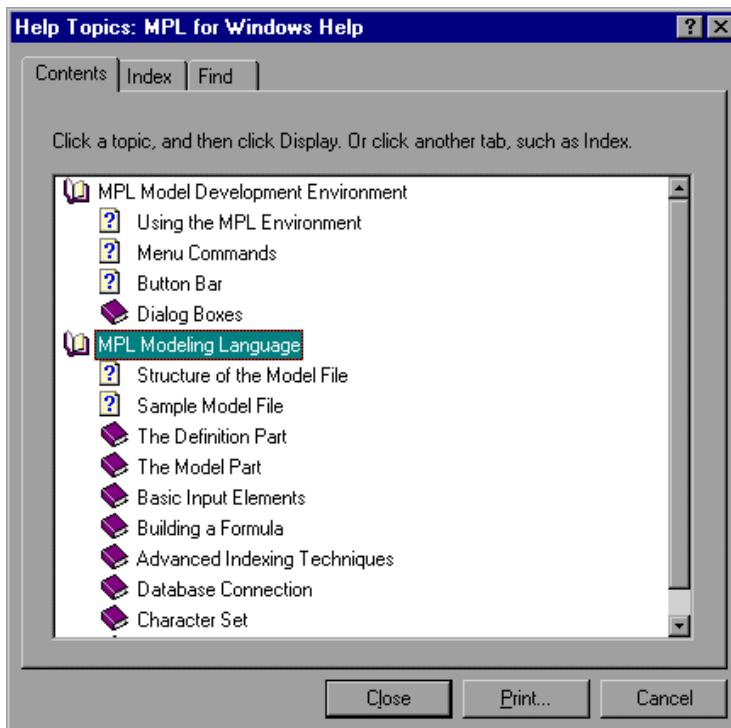


Figure 4.66: MPL for Windows Help Contents Window

You can double-click on any of the books to see all the topics available in that category or section. You can then continue opening books and browsing through the contents of the help until you locate a topic that looks useful.

You can print a topic by pressing the *Print* button at the bottom. If you want to print all the topics in a book, select it then press the *Print* button. Each topic will be printed on a separate page, though.

The Index tab

The *Index* tab of the help window enables the user to access a list of all the keywords in the help file or do a search for a specific keyword. The keyword list resembles a book index, with secondary entries indented beneath the primary entries.

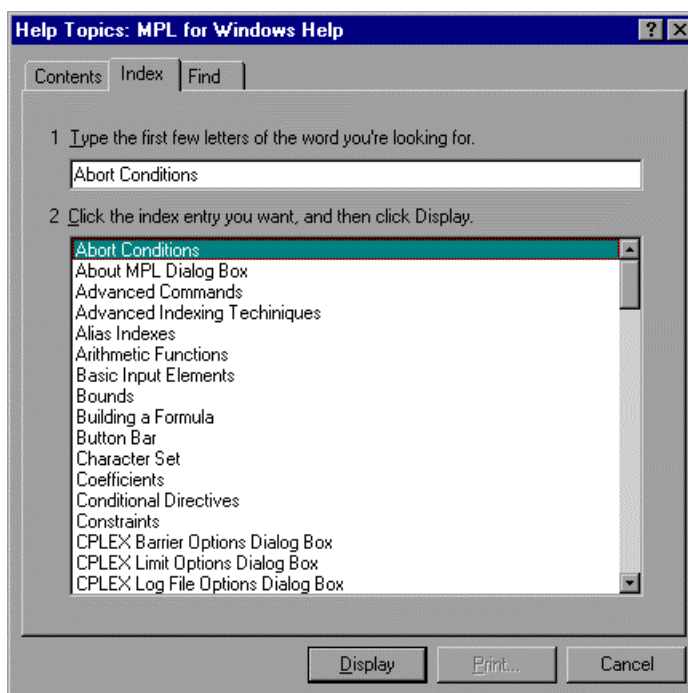


Figure 4.67: The Index Tab in the Help Window

As you type in a keyword, the list is automatically scrolled to match the characters typed against the primary entries in the list. If a word is not on the list, the user is taken to the word closest to where the word ought to be. Of course, you can still scroll through the keyword list and choose a word.

The Find tab

The *Find* tab enables the user to find a help topic by searching for a specific words or phrases in the text. The Find database is built the first time you select the *Find* tab through the *Find Setup Wizard*. The depth of the full-text search generated by the wizard is determined in the first wizard window. The recommended choice, *Minimize Database Size*, produces the smallest word list, while *Maximize Search Capabilities* gives the largest database and therefore the best search capabilities. You can further customize the search by pressing the *Options* button while in the Find tab.

About MPL for Windows

You can display the *About MPL for Windows* dialog box by choosing *About MPL* from the Help menu.



Figure 4.68: The About MPL for Windows Dialog Box

This dialog box shows you the current release number of **MPL**, copyright information, the maximum problem size, who the software is licensed to along with the serial number.

Part II Using the MPL Modeling System



PART III

THE MPL MODELING LANGUAGE

Chapter 5: Language Overview

Chapter 6: Defining Sets and Indexes

Chapter 7: Data for the Model

Chapter 8: Formulating the Model

Chapter 9: Building Formulas

Chapter 10: Advanced Indexing Techniques

Chapter 11: Database Connection

Part III The MPL Modeling Language



CHAPTER 5

LANGUAGE OVERVIEW

The **MPL** modeling language offers a natural algebraic notation that enables the model developer to formulate complex optimization models in a concise, easy-to-read manner. Among modeling languages, **MPL** is unrivaled in its expressive power, readability, and user-friendliness. The **MPL** modeling language was designed to be very easy to use with a clear syntax making the process of formulating models in **MPL** efficient and productive. **MPL** is a flexible language and can be used to formulate models in many different areas of optimization ranging from production planning, scheduling, finance, and distribution, to full-scale supply-chain optimization.

MPL is a very robust and stable software whose core modules have been through extensive use and testing over more than a decade. This assures that the **MPL** software is both reliable and dependable and can be used in mission-critical projects. Some of the more notable features of the **MPL** language include:

- Separation of the data from the model formulation.
- Import data from different data sources.
- Independence from specific solvers.
- Include files to make handling of large problems easier.
- Exclusion of parts of the model using conditional directives.
- Special Ordered Sets and Semi-continuous variables.
- WHERE/IF conditions to handle special cases.
- Readable and helpful error messages.

Sparse Index and Data Handling

One of the most important features of any modeling language is how it handles large amounts of data. What makes **MPL** so powerful is its ability to effectively handle very large sparse index and data sets. In addition, **MPL** has extensive flexibility when working with subsets of indexes, functions of indexes, and compound or multi-dimensional index sets. This allows the model formulator, for example, to index only over products that are made by each machine in a specific plant instead of having to go through all the products for all the machines and all the plants, which would be considerable slower.

Scalability and Speed

MPL can easily handle very large matrices with millions of variables and constraints. This is especially important when dealing with large supply-chain optimization models over multiple time periods that can grow very quickly. **MPL** has its own memory manager that can dynamically store models of any size, giving it a full scalability. The only limitation the model developer faces is how much memory is available on his or her machine. Typically, **MPL** uses only a few megabytes of memory per 10,000 variables, which puts a minimal additional burden on the machine capacity needed to generate and solve the model.

The matrix generation in **MPL** is extremely fast and efficient which is important since it contributes to the overall time needed to obtain the solution of the model. Maximal has over the years invested significant R&D efforts on continuing to improve the speed of the matrix generation. As a result, **MPL** can now run models with millions of variables and generate a matrix for them in less than one minute. This is very important, because if the model generation takes too long it can seriously add to the time needed to reach the solution even if the fastest optimization solvers are used. **MPL** provides the fastest and most scalable model generation capabilities available in a modeling system on the market today.

The **MPL** modeling language was developed with one key goal in mind; the input must resemble the notation people use to write their problem formulations on paper as closely as possible. In order to achieve that goal, advanced compiler and language parsing techniques normally reserved for writing high-level computer languages were used. The resulting language includes many features that can be of great assistance when formulating optimization models.

- Give each variable and constraint a meaningful name.
- Extend constraints over multiple lines.
- Place variables on both sides of constraints.
- Use arithmetic such as fractions and percentages.
- Factor out common coefficients using parentheses.
- Insert comments anywhere in the model.
- Use subscripted variables and data coefficients.
- Use macros for repetitive parts of the model.
- Use summations over vectors and arrays.

The input to **MPL** is a file that contains the model formulation. This file is a standard ASCII text file and can be created using the built-in model editor in **MPL**. Any text editor capable of working with text files can also be used.

5.1 Structure of the MPL Model File

The **MPL** model file is divided into two main parts; the definition and the model. In the definition part you define various items that are then used throughout the model. The model part, on the other hand, contains the actual model formulation. Each part is further divided into sections, which are as follows:

The Definition Part

TITLE	- The problem name.
INDEX	- Dimensions of the problem.
DATA	- Scalars, data vectors and files.
VARIABLES	- Decision variables.
MACRO	- Macros for repetitive parts.

The Model Part

MODEL	
MAX or MIN	- The objective function.
SUBJECT TO	- The constraints.
BOUNDS	- Simple upper and lower bounds.
FREE	- Free variables.
INTEGER	- Integer variables.
BINARY	- Binary (0/1) variables.
END	

None of the sections are actually required in **MPL**, but in order to have a valid optimization model you will need at least the objective function and the constraints. **MPL** allows you to place the sections in any order, but since any item must be declared before it is used in the model, the above order is used most of the time. Multiple entries of each section are allowed. The keywords *MODEL*, *SUBJECT TO* and *END* are optional, but used most of the time to aid readability. As an example of how simple the model in **MPL** can be, here is one tiny but still perfectly legal input:

```
MAX x + y ;
x + 2y < 10 ;
```

In this chapter each of the aforementioned sections is explained in full detail. On the next page is the **MPL** model file *'planning.mpl'* that we will use as an example.

Part III The MPL Modeling Language

```
{ Planning.mpl }

{ Aggregate production planning for 12 months }

TITLE
  Production_Planning;

INDEX
  product = 1..3;
  month   = (January,February,March,April,May,June,July,
            August,September,October,November,December);

DATA
  Price[product]           := (105.09, 234.00, 800.00);
  Demand[month,product]   := 1000 DATAFILE(demand.dat);
  ProductionCapacity[product] := 1000 (10, 42, 14);
  ProductionCost[product]  := (64.30, 188.10, 653.20);
  InventoryCost            := 8.8 ;

DECISION VARIABLES
  Inventory[product,month] -> Invt;
  Production[product,month] -> Prod;
  Sales[product,month]     -> Sale;

MACRO
  Revenues := SUM(product,month: price * Sales);
  TotalCost := SUM(product,month: InventoryCost * Inventory
                  + ProductionCost * Production);

MODEL

  MAX Profit = Revenues - TotalCost;

SUBJECT TO
  InventoryBalance[product,month] -> IBal :
  Inventory = Inventory[month-1] + Production - Sales;

BOUNDS
  Sales < Demand;
  Production < ProductionCapacity;
  Inventory[month=January..November] < 90000;
  Inventory[month=December] = 20000;

END
```

Figure 5.1: The Planning.mpl Model File

The Problem Title

The title section is used to give each problem a name which we strongly recommend. When you use a name, **MPL** places it in the generated file where appropriate, which helps you keep track of files and printouts. Your title section starts with the keyword *TITLE*, followed by the problem name. The name can be of any length but cannot contain spaces or other delimiters. You may place a semicolon after the name, although it is not necessary. If you omit the title, it defaults to the name *Problem*.

Example:

```
TITLE Production_Planning ;
```

The Definition Part

The Definition Part is used to define various items to be used later in the model. Here you can, for example, define subscripted variables, data to be imported from other programs and macros for repetitive parts of the model.

The first section, after the title, is the index section, starting with the keyword **INDEX**. This is where you define the indexes and the sets for the model. This will be covered in *Chapter 6: Defining Sets and Indexes*.

After the indexes, the next step is to setup the data for the model in the **DATA** section. You can specify both data defined directly in the model and also read in data from either external data files or databases. This will be covered in *Chapter 7: Data for the Model*.

Following the data section you normally define the variables for the model in the **DECISION VARIABLES** section. If you need to make the variables dependent on a certain sparse data vector you do this using the **WHERE** command. See *Chapter 8.1: Declaring Decision Variables* for further information.

The last section in the Definition Part is the **MACROS** section. The macros can be used to give complex expressions a distinctive name and then refer to these complex expressions throughout the model by the macro. See *Chapter 8.2: Defining Macros* for further information.

Each section in the definition part, except the problem title, can appear as often as you want. These sections normally appear in the order listed above, but they can appear in any order you choose. Please note though that each item must be defined before it is referred. In the following chapters each section will be discussed in detail.

The Model Part

The model part contains the actual problem formulation. Although the keyword *MODEL* is optional, it is useful to mark the beginning of the problem formulation.

The layout of the model is straightforward. The first step is the objective function. Either the keyword *MAX* or keyword *MIN* is used, depending on whether it is a maximization or minimization problem.

After the objective function, the next step is to list the constraints. The keyword *SUBJECT TO* is used as a separator between the objective function and the constraints.

The next step is to enter simple upper and lower bounds on variables if they are used. You use the keyword *BOUNDS* to mark the beginning of the bounds section.

Free variables are entered following the *FREE* keyword. Integer and binary variables are entered after the *INTEGER* and *BINARY* keywords, respectively. *SOS* sets of variables are entered after the *SOS* keywords. These last five sections (including bounds) are all optional and can be placed in any order.

The *END* keyword is optional and marks the end of the model. Everything that follows it is ignored.

5.2 Basic Input Elements

Numbers

You can enter numbers as real numbers or as integers; **MPL** uses them all as real (floating point) numbers. Scientific notation is *not* allowed. Integer numbers can be entered with a decimal point to the right. Numbers between -1.0 and 1.0 can be entered without a leading zero.

Examples of numbers recognized by MPL:

```
24
4.8565
13.
-.7854
```

Names

The names of variables and constraints can be of any length. A name begins with a letter followed by any combination of letters and digits. **MPL** is case sensitive by default, but you can change that in the *MPL Language Options* dialog box in the *Option* menu. See *Chapter 4.9: The Options Menu* for further information. You cannot use spaces in a name, but the underscore character '_' is allowed. For full list of characters allowed in variable names, refer to *Appendix A: Character Set*.

Examples of legal filenames:

```
x1
Inventory
```

An alternative way is to use double quotes ("). Then any character can be used, even spaces and other special characters. A quote symbol in quoted name is represented by two adjacent quotes. An empty name (name without any characters in it), which can be useful in name abbreviations, is written by two quotes with nothing between them (" ").

Examples of quoted names:

```
"Inventory in April"
Prod3 -> "P3"
x[foods] -> ""
```

Delimiters

MPL detects the end of a name when it encounters a delimiter. A delimiter is any of the following characters:

! \$ % & ' () * + , - . / : ; { } \ ^ _ ` { } | ~

Furthermore, the space character, the end of an input line, and any character in the ASCII range 0..31 are also treated as delimiters. This manual always specifies ASCII values with decimal (base 10) numbers.

White Space

MPL automatically skips over all white space characters when reading the input. White space characters are:

The space character: Spaces are often used to make the formulation easier to read. We recommend you use them extensively.

The control characters (ASCII range 0..31): You can use various control characters in the input for printing considerations without affecting **MPL**. This includes characters like Tab (ASCII 9) and FormFeed (ASCII 12).

Inserting Comments

A comment is a part of the input file that **MPL** does not process. Usually, comments are explanations of the formulation in plain words, but you can also use them to temporarily “remove” various parts of the model. Typically, you might want to hide a constraint, which you can later make visible again. Also, you can switch between sets of formulas or constraints. Conditional directives and include files (explained later in this chapter) are also often used for this purpose. You can insert comments anywhere in the input file. There are two basic types of comments:

Single-line comments: !

Single-line comments are used to add a comment of one line or a part of a line of text. When **MPL** finds the exclamation mark, it simply skips the rest of the line.

Block comments: { ... }

Block comments are used to add several lines or pages of text. When **MPL** reads the input, it skips everything inside the braces. For example to hide part of a formulation, perhaps only temporarily, you can add braces before and after the part in question. A block of text can even have comments contained within it; i.e., **MPL** can handle nested comments.

Include Files

Include files allows you, instead of storing the entire model in a single large file, to break it up into one or more files that are then included in the main model file. Place the include command on a line by itself, at the point where you wish the include file to be read. For example:

```
#include filename
```

In above example *filename* is the name of the file you wish to include. Include files can be nested up to 8 levels.

Include files make the construction of large models much easier and more reliable. The main advantages of using include files are:

- Several models can share the common part of a formulation. This could be a large objective function containing many prices, or several constraints that are the same in all the models. This sharing saves typing and simplifies the changes since only one file must be modified. This is particularly important when you are creating a group of models that work together.
- It is often convenient to group related parts of large problems together, and to put each group in a separate file to get better overview and make them easier to maintain.
- When you use more than one version of a constraint, it is better to place each version into a separate include file than to use many slightly different main files. With separate include file you can avoid repeating the parts of the model that stay the same. Conditional directives can be used for the same purpose. Refer to the next section on *Conditional Directives* on the next page for further information.

Conditional Directives

This feature in **MPL** is well known in computer programming. Conditional directives can be just as useful when developing linear programming models as when developing programs. They allow you to define special symbols called directives and then, based on these directives, to include or exclude some part of the model.

All directive commands take one line and must be the first item on that line. They start with the ‘#’ character followed by the command and the symbol in question, where applicable. You can nest the *#ifdef*'s up to any level.

Here is a list of the conditional directives that are allowed:

```
#define symbol      ! defines the symbol
#undef symbol       ! undefines the symbol
#ifdef symbol       ! include the following if symbol is defined
#ifndef symbol      ! include if symbol is not defined
#else              ! else on the last #ifdef
#endif             ! closes the last #ifdef
```

To illustrate this, let's say you have a problem with variables which can either be regular or integer variables. You can then use conditional directives to decide whether they are to be defined as integer in the model.

```
#define IntegerProblem
.
.
.
#ifdef IntegerProblem
    INTEGER
    x,y
#endif
```

In this example, you define the directive *IntegerProblem* in the beginning of the model formulation. Then, if you want to make the variables noninteger, you only have to delete or comment out the define for *IntegerProblem*.

Option Settings

In addition to setting options in the option dialog boxes (See Chapter 4, Section 4.9 Options Menu), **MPL** allows you to set certain options directly in the model file. This is accomplished by using the keyword *OPTIONS* anywhere in the model file, followed by a line for each option entry. For example, if you want to specify a different Input Directory for data files and change the model type to nonlinear you can enter the following:

```
OPTIONS
    DatafileInputDir="Data"
    ModelType=Nonlinear
```

The available option entries you can set in the model file are listed below. Each entry is listed with the name string that is used, the original name of the option, and then a short description.

PlainVarDefined: (*Plain Variables Must be Defined*) Specifies whether plain variables must be defined in the Decision Variable section of the model or can be introduced as they appear in the model. For larger models, requiring plain variables to be defined, can increase the maintainability of the model.

DatafileInputDir: (*Data Files Input Directory*) Selects the folder where **MPL** will search for input data files. The default is the current folder.

DatafileOutputDir: (*Data Files Output Directory*) Selects the folder where **MPL** will save output data files. The default is the current folder.

CheckDuplicateData: (*Check Sparse Data For Duplicate Entries*) Specifies whether sparse data files are checked for duplicate index entries. In some cases the user may want to read in a data file without receiving errors even if it has duplicate entries. When there are duplicate entries, the last entry in the file will be used by **MPL**.

UseQuickSort: (*Use quicksort for sparse data*) Specifies whether sparse data is sorted with *quicksort* after it is read in. It is normally faster to use *quicksort* but if there is an invalid entry, such as duplicates, in the data file **MPL** will not be able to pinpoint the problem line accurately.

ModelType: (*Default model type*) Specifies the default model type for **MPL** language.

Linear (1) Accept only models that are either linear or mixed integer.

Quadratic (2) Accept models that are quadratic, in addition, to the standard linear or mixed integer models.

Nonlinear (3) Accept models that are nonlinear. For example, models that have variables multiplied together or use nonlinear arithmetic functions such as LOG or EXP on the variables.

NameGenType: (*Name Generation*) Chooses the method **MPL** uses to generate names for vector variables and constraints.

Indexed (0) Generate names using the actual index elements.

Numeric (1) Generate names as numeric with prefix 'C' for variables and prefix 'R' for constraints.

Prefixed (2) Generate names as numeric with the prefix based on the vector name.

MaxVarLen: (*Max variable length*) Most LP solvers have a restriction on the length of variable names. Since **MPL**, in most cases, sends the matrix directly through memory to the solver, the variable names are normally not needed. If they are needed, the value set here helps you ensure that the variable names generated are within limits.

MaxSubLen: (*Max subscript length*) This value decides how many characters of indexes are retained in the generated variable name. This allows you to use long index names in the model, but keep variable names concise in the generated input file.

DatabaseType: (*Default*) Chooses which of the supported databases is the default for the **MPL** database connection. Valid entries are ODBC, Access, Excel, FoxPro, and Dbase.

DatabaseDirectory: (*Directory*) When the database type is either FoxPro or Dbase you can use this option to specify the directory where the database files are stored.

DatabaseODBC: (*ODBC Data Source*) When the database type selected is ODBC this option specifies which datasource, as defined in the ODBC control panel, should be used.

DatabaseAccess: (*Database File *.mdb*) When the database type selected is Access this option specifies which database should be used. Please note that if you use any special characters in the name you might have to enclose it with quotation marks.

For example, if you want to read data from the Access database *planning.mdb* enter the following options in the model file:

```
OPTIONS
DatabaseType=Access
DatabaseAccess="planning.mdb"
```

Part III The MPL Modeling Language

DatabaseExcel: (*Workbook File *.xls*) When the database type selected is Excel or Excel4 this option specifies which workbook should be used.

DatabaseUsername: (*User*) When you are working with databases that require a Username to log in, this option can be used to specify it.

DatabasePassword: (*Password*) When you are working with databases that require a Password to log in, this option can be used to specify it.

ExcelWorkbook: (*Workbook File *.xls*) Allows you to specify the default Excel Workbook filename when reading in Excel ranges.

For example, if you want to read Excel ranges from the Excel workbook file *cutstock.xls* enter the following options in the model file:

```
OPTIONS
  ExcelWorkbook="cutstock.xls"
```

ExcelSheetname: (*Worksheet name*) When the Excel range data is not on the first default worksheet, you can use this option to specify which sheet to read from.

ExcelSkipOverEmpty: When reading indexes from an Excel range, the default is to skip empty cells. You can use this option to turn this *Off* causing **MPL** to stop reading at the first empty cell.

CHAPTER 6

DEFINING SETS AND INDEXES

The index section is used to define the *domains* of the model. The indexes are then used throughout the model when objects like subscripted variables and data coefficients, summations and structured constraints are used. Indexes encapsulate the problem dimensions and make it easy to quickly adjust the problem size.

There are two basic types of indexes; numeric and named. The numeric indexes are used to give each subscript item a numerical value. The named indexes are used to give each subscript item a descriptive name.

6.1 Numeric Indexes

Numeric indexes are specified within a range by entering the lowest value and the highest value separated by two periods. Data constants and any legal integer formula, including integer functions, can be used when specifying the numbers. Both numbers must be non-negative and the second must be larger than the first. The following example defines an integer index with three elements:

```
INDEX
  product := 1..3;
```

After you have defined an index you can then use it to define data, variable, and constraint vectors. For example, the following example would define a variable vector named *Ship* that contains three elements:

```
VARIABLES
  Ship[product];
```

This definition will generate three variables (one for each product), named *Ship1*, *Ship2*, and *Ship3*.

The name length of most LP solvers is limited to eight characters. Therefore, only the minimum number of characters are used, for each numeric index, is normally used when building variable and constraints names in **MPL**.

It is not always desirable to have the default subscript length for all indexes in the model. You can set a different length for each index by following the index definition with a colon and a new length. Here is an example:

```
INDEX
  i := 1..5 : 2;
```

This definition sets the subscript length for the *i* index to 2.

6.2 Named Indexes

Named indexes are specified by listing the subscript elements within parenthesis. Each item in the list has a separate name that is then used when referring to the subscripts. This is a convenient way of assigning meaning to indexes and the vectors that use them. Here is an example:

```
month := (January,February,March,April,May,June,July,
         August,September,October,November,December);
```

In most cases **MPL** sends the matrix to the LP solver directly through memory, which means that the names are not used. But sometimes, there is a need to generate either a log file or an input file where the names would become useful. Because the name size of most LP solvers are limited to eight characters, only the first three characters, for each named index, are normally used when building variable and constraints names in **MPL**. The default number of characters used can be set in the *Max Subscript Length* option in the *MPL Language* dialog box in the *Options* menu. See *Chapter 4.9 The Options Menu* for more information.

To make sure there are no name conflicts, keep at least the first three characters of the named subscripts distinct. If that is difficult, you can abbreviate the names by using the *becomes operator* ‘->’. To use the *becomes operator* insert ‘->’ after the subscript list and place the other list with shorter version of the names beneath. These abbreviated names are then used when generating the input file. Here is an example:

```
City := (SanDiego, LosAngeles, SanFrancisco)
        -> (SD, LA, SF);
```

It is not always desirable to have the same subscript length for all indexes in the model. Therefore you can use a different length for each named index by following the index definition with a colon and a new length. Here is an example:

```
INDEX
month := (January,February,March,April,May,June) : 4;
```

This definition sets the subscript length for the *month* index to 4.

If the index contains numeric values, but you do want them treated as a named index you can do so by adding the keyword *NAMED* in front of the *INDEX* keyword. This is, for example, necessary when you have product numbers that are all numeric, but you do not want them sorted according to their numerical values.

6.3 Alias Indexes

In **MPL** you can give a defined index another name. This is called an *alias* index and is used to facilitate repeated use of the same index in a vector, for example, in transportation problems. For instance:

```
INDEX
  location := (A, B, C, D);
  source   := location;
  dest     := location;
```

This expression defines the *source* and *dest* indexes as an alias of the *location* index and allows you, for example, to define the following variable:

```
DECISION VARIABLES
  Ship[source,dest];
```

6.4 Circular Indexes

When working with indexes that represent time periods you sometimes encounter situations where you need to offset the index subscript value. For example, when you want to use a data entry from the previous month, **MPL** allows you to do this by using a offset value with the subscript such as in *Inventory[month-1]*.

When the offset value goes outside the defined range for the index, **MPL** by default omits that vector entry. By defining an index as circular, you specify that when the offset value goes out of range, it takes values instead from the opposite end.

Index are defined as circular by entering the keyword *CIRCULAR* after the index definition. For example:

```
INDEX
  day      := (mon, tue, wed, thu, fri, sat, sun) CIRCULAR;
  month    := 1..12 CIRCULAR;
```

6.5 Subsets of Indexes

Sometimes, you want to use only a few elements or *subset* of an index. **MPL** allows you to declare a new index that selects certain elements of another underlying index. For example, assume you have defined the following indexes:

```
INDEX
  i := 1..10
  m := (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);
```

You can then use them to define these indexes as subsets of the original indexes:

```
j[i] := (2,3,5);
HolidayMonth[m] := (Apr, Jul, Aug);
RepairMonth[m] WHERE (m >= Apr) AND (m <= Jun);
```

Here the underlying index for j is i and for *HolidayMonth* is m . You can also define an index as a list of numbers, i.e. an enumeration of numbers instead of names. **MPL** treats this kind of index as a subset index without an underlying index. For example:

```
k := (0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75);
```

6.6 Function Indexes

In some cases, when working with indexes you want to have the ability to be able to map certain index elements to elements of another index. In **MPL** this is done with *function indexes*.

For example, you are creating a model that schedules trucks to certain routes, but their current location is not where each route starts. Therefore, in the model we create two function indexes. One that maps each truck to the city where they are currently located and the other where we map each route to the city of departure. In **MPL** this can be formulated as follows:

```
INDEX
truck := (T1, T2, T3, T4, T5, T6);
route := (R1, R2, R3, R4, R5, R6);

city := (Atlanta, Chicago, Dallas, Denver, NewYork);
Current[city];
Depart[city];

CurrentLoc[truck].Current :=
(T1,Atlanta, T2,Atlanta,
 T3,Atlanta, T4,Atlanta,
 T5,Chicago, T6,Chicago);

DepartLoc[route].Depart :=
(R1,Chicago, R2,Dallas,
 R3,Denver, R4,Denver,
 R5,NewYork, R6,NewYork);

DATA
Distance[Current, Depart] := ( 0, 717, 783, 1406, 886,
                               717, 0, 937, 1023, 807,
                               783, 937, 0, 794, 1576,
                               1406, 1023, 794, 0, 1785,
                               886, 807, 1576, 1785, 0);

VARIABLES
Assign[truck, route];
```

Now we can create an objective function that minimizes the total distance for each assigned truck route as follows:

```
MIN TotalDistance =
SUM(truck,route: Distance[CurrentLoc.Current, DepartLoc.Depart]
    * Assign[truck, route]);
```

6.7 Set Operations on Indexes

When working with subsets of indexes, you sometimes want to create new indexes based on previous ones. **MPL** allows you to use standard set operations such as *difference*, *not*, *union*, and *intersection* to help define new indexes.

The *set difference operation* is specified by using the minus (-) symbol between two indexes. It will subtract from the first index all the elements that are in the second index and create a new index containing the remaining elements.

The *not operation* on index is specified by placing the keyword *NOT* in front of the index. It will select all the elements from the parent index that are not defined in the given index.

The *set union operation* can be specified by placing either the plus symbol (+), the *OR* keyword or the *UNION* keyword between two indexes. It merges all the elements from both of the indexes into one large index.

The *set intersection operation* is specified by placing either the *AND* keyword or the *INTERSECTION* keyword between two indexes. It creates a new index that contains only the elements that are defined in both of the indexes.

Example of set operations:

```
INDEX
plants := (NewYork, Chicago, London, Paris);

OpenPlants[plants] := (NewYork, London);
EuropePlants[plants] := (London, Paris);

! Difference
ClosedPlants[plants] := plants - OpenPlants;
                    := (Chicago, Paris)

! Not
USPlants[plants] := NOT EuropePlants;
                 := (NewYork, Chicago)

! Union
OpenOrEurope[plants] := OpenPlants OR EuropePlants;
                    := (NewYork, London, Paris)

! Intersection
OpenAndEurope[plants] := OpenPlants AND EuropePlants;
                     := (London)
```

6.8 Multi-dimensional Index Sets

Often, when formulating models with multiple indexes, some of the indexes are connected together. For example, if your model has one index defined as *plant*, and another index defined as *machine*, you might want to be able to create an index specifying which machines are available in which plants. This can be done in **MPL** using *multi-dimensional index sets*. They are defined in a similar way to normal subset indexes, but instead of only a single domain index being placed inside the brackets, you enter a list of domain indexes separated by commas.

```
INDEX
  plant      := (Atlanta, Chicago, Dallas);
  machine    := (Grind, Drill, Press);

  PlantMachine[plant,machine] :=
      (Atlanta.Grind,
       Chicago.Drill,
       Chicago.Press,
       Dallas.Grind);
```

In the above example we created a two-dimensional index *PlantMachine* that contains, for each plant, the machines that are available.

When referring to multi-dimensional indexes we sometimes call it the *parent* index and the indexes it is derived from the *domain* indexes. When specifying which element pairs to include in the parent index you can, as in the above example, list all the elements you want to include, with each domain index element separated with a period so long as they are not numeric indexes. Alternatively, you can also separate each domain index element with a comma. For clarity, you can also group each parent index element with parentheses.

```
PlantMachine[plant,machine] :=
      ((Atlanta, Grind),
       (Chicago, Drill),
       (Chicago, Press),
       (Dallas, Grind));
```

Instead of listing all the elements manually, you can also use the *WHERE* command to specify which elements are included in the multi-dimensional index. We will now, on the following page, show several examples on how the *WHERE* command can be used when defining multi-dimensional indexes.

Chapter 6 Defining Sets and Indexes

In the example below we want to create an index *WinterRepair* that contains all the months in the *Repair* index which are during winter between *November* and *March*.

```
INDEX
  month := (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);
  Repair[month] := (Jan, Mar, Jun, Oct, Nov);
  WinterRepair[month] :=
    Repair[month] WHERE (month <= Mar) OR (month >= Nov);
!
```

In the next example we first define two indexes *node1* and *node2* that contains nodes that are in a network and then use the *WHERE* command to create a two-dimensional index *arcs* that connects together all the nodes except those that are the same.

```
INDEX
  node1 := (n1, n2, n3, n4);
  node2 := node1;
  arcs[node1, node2] WHERE (node1 <> node2);
```

When defining indexes you can also use the *WHERE* command to make the index dependent on the values of a data vector. In the example below, we want to create an index that contains the routes between the plants and the warehouses that have shipping costs below *2400*.

```
INDEX
  plant := (Atlanta, Chicago, Memphis);
  warehouse := (Pittsburgh, Charlotte, Pittsburgh);
DATA
  ShipCost[plant, warehouse] := (1200, 3000, 2300,
    1800, 4200, 3300,
    2700, 2100, 1900);
INDEX
  BestRoutes[plant, warehouse] WHERE (ShipCost <= 2400);
!
```

6.9 Reading Index From External File

You can use data files when specifying the elements for indexes. Instead of the usual list of elements, enter the keyword *INDEXFILE* followed by a filename inside parentheses. If you use any characters in the filename, such as '-' and '#', that have special meaning to **MPL**, you can enter them by including the filename in double quotes (").

```
INDEX
  product := INDEXFILE("product.dat");
```

The file is a free-format text file, in which the elements of the index are read in the order they appear. Separate the elements with commas or spaces. Everything inside curly braces and following an exclamation mark is treated as a comment.

Sometimes, the file you want to read the index elements from contains other columns of data. **MPL** allows you to specify which column is chosen for the index by adding a comma and a column number inside the parenthesis. For example, if the *product* index is stored in the first column of the *product.dat* file you would enter:

```
INDEX
  product := INDEXFILE("product.dat", 1);
```

Sample index file with one column for the *product* index and two data columns is shown below:

```
prod1, 123.55, 245.90
prod2, 347.00, 365.50
prod3, 223.22, 389.60
prod4, 439.50, 445.50
```

Reading subset and multi-dimensional indexes from an index file is done in the same way as for normal indexes. The only difference is that each entry in the index file needs to contain the elements for each of the domain indexes. The entry in the model file is the same way as before.

```
INDEX
  FactoryMachine[factory,machine] := INDEXFILE("factmach.dat");
```

Here is a sample index file with two columns, one for factory and one for machine:

```
fac1, mach1,
fac2, mach2,
fac2, mach3,
fac3, mach4,
fac3, mach5,
```

6.10 Import Index from Excel Spreadsheet

MPL allows you to import the elements for an index directly from a Excel spreadsheet. In the *INDEX* section, where you define the index, enter the keyword *EXCEL RANGE* after the assignment symbol (*:=*) followed by parentheses containing the Excel workbook filename and the Excel range name you want to import from.

```
INDEX
cuts := EXCEL RANGE ("Cutstock.xls", "CutsRange");
```

In the above example, **MPL** will open the Excel spreadsheet *Cutstock.xls*, locate the *CutsRange*, and then read in the entries for the index *cuts*.

To make sure there are no name conflicts when sending the problem to the solver, keep at least the first three characters of the named subscripts distinct. If that is difficult, you can abbreviate the names by importing another range from the spreadsheet containing shorter names. To import the short name range enter the keyword *BECOMES* followed by the range name inside the parenthesis. Here is an example:

```
INDEX
cuts := EXCEL RANGE ("Cutstock.xls", "CutsRange", BECOMES "CutsShort");
```

It is not always desirable to have the same subscript length for all indexes in the model. Therefore you can use a different length for each named index by following the index definition with a colon and a new length. For example:

```
INDEX
cuts := EXCEL RANGE ("Cutstock.xls", "CutsRange"):4;
```

This definition sets the subscript length for the *month* index to 4.

The *EXCEL RANGE* command will read every cell in the range by default until an empty cell is encountered. In some cases, you will want to read only a specific column from the range. **MPL** allows you to specify which column to read by entering a comma and the column number after the range name. For example:

```
INDEX
cuts := EXCEL RANGE ("Cutstock.xls", "CutsTable", 1);
```

This will read only the first column of the *CutsTable* range.

Also note, that if you are reading multiple indexes and data vectors from the same Excel spreadsheet, you can omit the workbook filename on all entries after the first one.

6.11 Import Index from Database

MPL allows you to import the elements for an index directly from a database. In the *INDEX* section, where you define the index, enter the keyword *DATABASE* after the assignment symbol (*:=*) followed by parentheses containing the table name and the column/field name you want to import from.

```
INDEX
  depot := DATABASE("Depots", "DepotID");
```

In the above example, **MPL** will open the database table *Depots*, locate the column *DepotID*, and then read in the entries for the index *depot*. In most cases the imported indexes are the key fields for the table.

To make sure there are no name conflicts when sending the problem to the solver, keep at least the first three characters of the named subscripts distinct. If that is difficult, you can abbreviate the names by importing another column from the database table containing shorter names. To import the short name column enter the keyword *BECOMES* followed by the column name inside the parenthesis. Here is an example:

```
INDEX
  depot := DATABASE("Depots", "DepotName", BECOMES "DepotID");
```

It is not always desirable to have the same subscript length for all indexes in the model. Therefore you can use a different length for each named index by following the index definition with a colon and a new length. For example:

```
INDEX
  depot := DATABASE("Depots", "DepotID"):4;
```

This definition sets the subscript length for the *month* index to 4.

In some instances you do not want to create the index with all the elements that are in the table. In that case, you can enter the keyword *WHERE* followed by a condition on one of the columns. Here is an example:

```
INDEX
  depot := DATABASE("Depots", "DepotID", WHERE Region="NorthWest");
```

For further information on importing indexes please refer to *Chapter 11.1: Import Indexes from Database*.

CHAPTER 7

DATA FOR THE MODEL

In the data section you specify the data coefficients to be used in the model. These can be scalars, vectors, and even arrays of multiple dimensions. By defining the data coefficients separately, the actual model is free of numerical data and thus easier to maintain. Furthermore, by using the import feature you can store the data in a separate file and retrieve it when generating the input file.

Each data coefficient is given a separate name that is used throughout the model. There are two different types of data objects that **MPL** recognizes; data constants or scalars that are mainly used to aid readability and make the model easier to maintain; and data vectors which are used when the coefficients come in lists or tables of numerical data. The following sections describe how both are specified.

7.1 Data Constants

Data constants or scalars are mainly used to aid readability and make the model easier to maintain. You can specify them in two different ways: by assigning them a value in the model or interactively by prompting the user for the value when the model file is read.

To use the interactive method, place a question mark following the coefficient value.

Example:

```
DATA
  InventoryCost      := 8.8;
  MaximumInventory  := 1200?;
```

Here the *InventoryCost* is given the value 8.8 which will be used whenever the constant is referred to in the model.

The value for *MaximumInventory* is prompted at runtime as in the *Data Request* dialog box like the one shown below in Figure 7.1. It will then be used in the model like any other constant.



Figure 7.1: Data Constant Entered at Run-time

Named data constants can be used almost anywhere in the model. This can be used to help make the model more dynamic and easy to change. Here are some examples:

```

DATA
  NrOfYears   = 10 ?;
  December    = 12;
  HolidayMonth = 7;
INDEX
  i := 1..NrOfYears;
  j := 1..December;
VARIABLES
  Inventory[i,j];
  Production[i,j];
  .
  .
BOUNDS
  Inventory[i,December] > 200;
  Production[i,HolidayMonth] < 100;
END

```

The value of the named data constant *NrOfYears* is prompted at run-time using the default value of 10. It is then used to define the index *i*. The *HolidayMonth* constant is used to enter a fixed month for the *Production* variable.

Like with indexes and data vectors, **MPL** allows you to import data constant entries from datafiles and Excel spreadsheets. Please note, that since data constants are not indexed they cannot easily be imported from databases. To read from a datafile just enter the *DATAFILE* keyword followed by the filename in parentheses. To read from an Excel spreadsheet, enter the *EXCEL RANGE* keyword followed by the workbook filename and the cell range in parentheses. For example:

```

DATA
  NrOfYears   = DATAFILE("years.dat");
  December    = 12;
  HolidayMonth = EXCEL RANGE("worksched.xls", "B5");

```

7.2 Data Vectors

Data vectors are used when the coefficients come in lists or tables of numerical data. They can be specified as lists of numbers in the model file or retrieved from an external file. With both methods you must specify the name of the data vector and the indexes that define the domain of the vector. To specify the indexes used, list the index names in brackets immediately after the vector name and separate them with commas.

The simplest way to define the values of the data vector is to enter them as a list of numbers directly after an assignment symbol. Surround the list with parentheses and separate each number by either space, comma, or both. To enter a zero value, you can alternatively enter only the comma to reduce typing.

```
DATA
  Price[product]      := (105.09, 234.00, 800.00);
  ProdCapacity[product] := 1000 (105, 42, 14);
```

Here *price* is a vector over one dimension *product*. Because there are three products, a list of three numbers is given. There must be enough numbers to satisfy the size of the vector, but additional values are allowed and simply ignored.

In *ProductionCapacity* 1000 is used as a multiplier. The actual values for the vector will be (105000, 42000, 14000).

Alternatively, you can put a multiplier in front of the parentheses and it is then applied to each number in the list. This feature reduces typing and increases the models readability.

As an additional feature, when defining a data vector, constant formulas can be used instead of just numbers for the elements. This includes features such as products, fractions, and arithmetic functions. Here is an example:

```
A[i] := (2, -4+3, 2*SQR(3)+2, 1/(2+1), last(i));
```

When specifying tables of two or more dimensions, you can use just one set of parentheses for the whole list or use new parentheses for each dimension.

```
C[i,j] := (4,2,5,6,      or   C[i,j] := ((4,2,5,6),
          8,3,8,4,      (8,3,8,4),
          3,6,1,0);     (3,6,1,0));
```


7.3 Sparse Data Vectors

MPL allows you to choose between a dense or sparse storage for the specification of data vectors. **MPL** is typically a little faster when it processes dense data as compared to sparse data, but storage is needed for every element. Therefore, when you are working with large matrixes containing relatively few nonzero elements, sparse storage become significantly more effective.

To define a sparse data vector, you surround the list of elements with brackets instead of the usual parentheses. Inside the brackets you list each nonzero element preceded by its subscript:

```
A[i] := [2: 4.0, 5: 3.0, 6: -4.0] ;

ProdCost[plant, machine, product] := [
    p1, m11, A1, 73.30,
    p1, m11, A2, 52.90,
    p1, m12, A3, 65.40,
    p1, m13, A3, 47.60,

    p2, m21, A1, 79.00,
    p2, m21, A3, 66.80,
    p2, m22, A2, 52.00,

    p3, m31, A1, 75.80,
    p3, m31, A3, 50.90,
    p3, m32, A1, 79.90,
    p3, m32, A2, 52.10,

    p4, m41, A1, 82.70,
    p4, m41, A2, 63.30,
    p4, m41, A3, 53.80];
```

In most cases when you have larger data sets the data itself is normally not specified in the actual model file but instead either is stored in an external datafile or a database table. See *Chapter 7.5 Reading Data from External Datafiles* and *Chapter 7.6 Import Data from Database* for further information.

MPL also allows you to separately specify whether the data will be stored sparse or dense in memory, independently of how it is defined in the model. Simply put either of the keywords *DENSE* or *SPARSE* in front of the section keyword *DATA*.

7.4 Data Arithmetic

In **MPL** previously defined data vectors can be used to construct new data vectors. You construct a data vector by placing an arithmetic formula, instead of the usual list of data elements, in the *DATA* section. Here are some examples:

```
DATA
  A[i,j] := ...
  B[i]    := ...
  C[i,j]  := A + B/3;
  Total   := SUM(i,j: A);
```

If the formula starts with a data vector, its type is used to decide whether the resulting vector is dense or sparse. If the formula is a summation, the resulting vector is always dense. If the formula does not start with a data vector or a summation or you need to override the default type, you can use the keywords *DENSE()* and *SPARSE()* to specify the type. For example, to specify the *C[i,j]* vector as stored sparse, use:

```
C[i,j] := SPARSE(A + B/3)
```

7.5 Reading Data From External Files

MPL supports reading in data from external text files. Text files are mainly used when the data for the model is stored locally, generated by other programs or by running SQL queries from a database. The data can be read either in a dense format, where all numbers are specified, or in a sparse column format, where only the numbers that have non-zero values are included. **MPL** can also read single scalar numbers for data constants. The data file can be any free-format text file, which gives the developer the ability to read the data from different sources without first requiring a conversion to a standard format. Reading data from text files is one of the fastest ways available to import data into **MPL**.

When reading data from a text file, use the keyword *DATAFILE* followed by a filename inside parentheses. If you use any characters in the filename, such as '-' and '#', that have special meaning to **MPL**, you can enter them by including the filename in double quotes (").

```
Demand[product,month] := 1000 DATAFILE("demand.dat");
```

In our example, the demand vector has 36 values, 12 months times 3 products, which are stored in the file *demand.dat*. Each number in the file is multiplied by 1000 as it is read. The following is an example of a data file with demand values for each product and month.

```

{ Demand.dat }
! Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
! -----
  50, 60, 70, 80, 90, 100, 110, 120, 120, 120, 110, 100
  24, 30, 36, 42, 48, 52, 50, 48, 44, 40, 36, 32
   5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 14, 13

```

The file is a free-format text file, in which numbers are read in the order they appear. Separate the numbers with commas or spaces and make sure there are enough numbers to satisfy the size of the vector. In order to use comments in the *DATAFILE* insert everything inside curly braces for multi-line comments and following an exclamation mark for single line comments. See *Chapter 5.2: Basic Input Elements* for more information on inserting comments.

MPL allows you to read multiple data vectors and simple data constants from the same datafile. Simply group them together in the data section and give the same filename inside the parentheses. **MPL** will then continue reading the values from where it left off from the previous definition.

Sometimes, the data you need to read is sparse, meaning that not all the index entries have values. In this case, to read the sparse data from an external file, use the keyword *SPARSEFILE* instead of *DATAFILE*. **MPL** then expects both the subscripts and the values for each entry in the given file. Here is an example that shows how you can read a sparse datafile into a data vector:

```
ProdCost[machine, product] := SPARSEFILE("prodcost.dat");
```

The file '*produce.dat*' could look something like:

```

mach1, prod1, 73.30,
mach1, prod2, 52.90,
mach1, prod3, 65.40,
mach1, prod4, 47.60,
mach2, prod1, 79.00,
mach2, prod3, 66.80,
mach3, prod2, 52.00,
mach3, prod3, 53.80

```

If the file you want to read the data elements from contains other columns of data, **MPL** allows you to specify which column is chosen, by adding a comma and a column number after the filename inside the parentheses.

```
ProdCost[machine, product] := SPARSEFILE("prodcost.dat", 4);
```

This will read the production cost values from the fourth column of the data file. Here is a sample of the datafile *produce.dat*:

```

mach1, prod1, 450, 73.30,
mach1, prod2, 500, 52.90,
mach1, prod3, 350, 65.40,
mach1, prod4, 425, 47.60,
mach2, prod1, 355, 79.00,
mach2, prod3, 500, 66.80,
mach3, prod2, 344, 52.00,
mach3, prod3, 250, 53.80

```

7.6 Import Data from Excel Spreadsheet

Optimization in *Microsoft Excel* has over the years become popular because of the built-in solver in *Excel*. Spreadsheet optimization allows users to create models that are easy to use, enabling the user to quickly update the data and solve the model. Spreadsheets are efficient at handling and managing two-dimensional dense data (rows and columns) and single scalar values.

Excel has the *Visual Basic* programming language built in, which enables easy pre- and post-processing of the data. The solution from the optimization can then be used to update the spreadsheet, and to create graphs that visually represent the solution.

Excel provides some excellent user-interface capabilities for optimization models. **MPL** builds on this by offering the ability to import and export data directly from *Excel* ranges. This allows the developer to use an *Excel* spreadsheet for the user interface and data manipulation while using **MPL** to specify the model formulation. The model can then be solved with any solver supported by **MPL** with no limits on the size, speed or robustness of the solution.

Furthermore, **MPL** is available as a part of the **OptiMax 2000 Component Library**, which is specially designed to embed optimization models into application programs, such as *Excel*. **OptiMax 2000** allows **MPL** models to be linked directly with the *Visual Basic for Applications* language in *Excel* enabling the developer to create large-scale optimization models that can be solved directly from an *Excel* spreadsheet. Please contact Maximal Software for more details.

In the *DATA* section, where you define the data vector, enter the keyword *EXCEL RANGE* after the assignment symbol (*:=*), followed by parentheses containing the *Excel* workbook filename and the *Excel* range name you want to import from.

To give an example of how data is imported from an *Excel* range, here is an **MPL** statement that is used to read in the pattern cuts generated for a cutting stock model:

```
DATA
  CutsInPattern[patterns, cuts] := EXCEL RANGE("Cutstock.xls", "Patterns");
```

In the above example, **MPL** will open the *Excel* spreadsheet *Cutstock.xls*, locate the range *Patterns*, and then retrieve the entries for the **MPL** data vector *CutsInPattern*.

In some cases, you will want to read only a specific column from the range. **MPL** allows you to specify which column to read by entering a comma and the column number after the range name.

The *EXCEL RANGE* command is used when the data is stored in a dense format in the spreadsheet. In some instances it is better to store the data in a sparse column format where the first columns store the values for the indexes followed by columns containing the data values. In this case you use the *EXCEL SPARSE* command to specify that the data is to be read in a sparse format.

For example:

```
DATA
  CutWidths[cuts] := EXCELSPARSE("CutsTable", 2);
  CutDemand[cuts] := EXCELSPARSE("CutsTable", 3);
```

In this example the Excel range *CutsTable* contains three columns. The first column stores the values for the index *Cuts* followed by two columns containing the widths and the demand for each cut. The column number 2 given means the second column in the range while 3 specifies the third column in the range.

If the Excel range contains the data as an *Excel List* there will be an extra row at the top of the range that contains the name of each column. **MPL** allows you to skip that row automatically by using the *EXCELLIST* command instead of the *EXCELSPARSE* command.

Also note, that if you are reading multiple indexes and data vectors from the same Excel spreadsheet, you can omit the workbook filename on all entries after the first one.

7.7 Import Data from Database

MPL allows you import the elements for a data vector directly from a database. In the *DATA* section, where you define the data vector, enter the keyword *DATABASE* after the assignment symbol (*:=*), followed by parentheses containing the table name and the column/field name you want to import from.

```
DATA
  FactDepCost[factory,depot] := DATABASE("FactDep","TrCost");
```

In the above example, **MPL** will open the database table *FactDep*, locate the columns *TrCost*, *FactID*, and *DepotID*, and then read in the entries for the data vector *FactDepCost*.

In some instances you do not want to read all the data elements that are in the table. In that case, you can enter the keyword *WHERE* followed by a condition on one of the columns. Here is an example:

```
DATA
  FactDepCost[factory,depot] :=
    DATABASE("FactDep","TrCost" WHERE Region="NorthWest");
```

In this example we only want to read the transportation costs from the *FactDep* table where the *Region* column contains the entry *NorthWest*.

For further information on importing data vectors, please refer to *Chapter 11.2: Import Data Vectors from Database*.

CHAPTER 8

FORMULATING THE MODEL

8.1 Declaring Decision Variables

The decision variables are the elements under control of the model developer and their values determine the solution of the model. Decision variables in **MPL** are defined as they come in the input file. The name can be of any length. It starts with a letter, with the remainder consisting of letters and digits. **MPL** is case sensitive by default so you need to be careful to distinguish between upper and lower case letters. If you want **MPL** not to be case sensitive you can change the default by selecting *MPL Language Dialog Box* in the *Options* menu. See *Chapter 4.9: The Options Menu* for further details.

There are two types of decision variables: plain variables and vector variables also called subscripted variable. A plain variable is setup as a single column in the matrix that is sent to the LP solver. A vector variable, on the other hand, is setup as range of columns in the matrix.

The *DECISION VARIABLES* section is mainly used to define subscripted or vector variables, but plain variables can also be defined here. Vector variables must be defined before they are used in the model, but with plain variables it is optional. Plain variables can either be defined in the Decision Variable section or be introduced as they come into the model. It is recommended for larger models to define plain variables in the Decision Variable section in order to decrease the chance of errors occurring in the model. If you want to require plain variables to be defined you set the *Plain Variables must be defined* option in the *MPL Language Dialog Box* in the *Options* menu.

Variable Names

For each vector variable, enter the name and then specify what indexes are to be used by listing them inside brackets just after the name. For plain variables, enter just the name. You can use an optional semicolon to separate each variable definition.

Because variable names in most LP solvers are limited to eight characters and the subscripts for vector variables have to be included, only the first few characters are used when variable names are generated for solvers by **MPL**. To make sure there are no name conflicts, keep the first few characters distinct or use the becomes operator ‘->’ to enter a shorter version of the name. This feature allows you to use long, descriptive names in your **MPL** model, and at the same time meet your LP package requirements for short names.

```
DECISION VARIABLES
  Inventory[product,month]  ->  Invt;
  Production[product,month] ->  Prod;
```


The preceding definition results in the generation of 108 variables (3 times 36). As an example of how these variables are generated, here is a list of all the *Inventory* variables:

Invt1Jan	Invt2Jan	Invt3Jan
Invt1Feb	Invt2Feb	Invt3Feb
Invt1Mar	Invt2Mar	Invt3Mar
Invt1Apr	Invt2Apr	Invt3Apr
Invt1May	Invt2May	Invt3May
Invt1Jun	Invt2Jun	Invt3Jun
Invt1Jul	Invt2Jul	Invt3Jul
Invt1Aug	Invt2Aug	Invt3Aug
Invt1Sep	Invt2Sep	Invt3Sep
Invt1Oct	Invt2Oct	Invt3Oct
Invt1Nov	Invt2Nov	Invt3Nov
Invt1Dec	Invt2Dec	Invt3Dec

Even though you can use fairly long names for variables in **MPL** you might want to use longer names or a description for documentary purposes. **MPL** allows you to do that by following the definition with the keyword *IS* and a text description. If the description contains spaces you will need to enclose it within quotation marks.

```
DECISION VARIABLES
  Production[product,month]
  IS "Production of each <product> per <month>";
```

Integer Variables

If some of the variables in the model are integer you can define them by putting the keyword *INTEGER* in front of the keyword *VARIABLES*. For binary variables use the keyword *BINARY*.

```
INTEGER VARIABLES
  Production[product,month];
```

MPL allows you also to define integer and binary variables at the end of the model. For more information see *Chapter 8.6 Integer and Binary Variables*.

Initial Values for Variables

When working with nonlinear models it is sometimes useful to be able to specify the initial values for the variables. In **MPL** you can enter the keyword *INITIAL* after the definition of the variable followed by a constant value or a data vector containing the initial values.

```
DECISION VARIABLES
  Production[product] INITIAL InitProd[product];
```

For more information on reading nonlinear models in **MPL**, see *Option Settings* in *Chapter 5.2 Basic Input Elements*.

Where Conditions on Variables

Sometimes you want to be able to limit the number of variables defined in a vector dependent on a data vector. In this case you can use a *WHERE data condition* following right after the definition of the vector. This will result in only those variables being created where the condition is met.

```
DECISION VARIABLES
  Production[product,month]
  WHERE (Demand[product,month] > 0);
```

Export Variable Values to Data Files

After optimizing the problem, **MPL** can export the variable values to separate data files which can be used to report the solution back to the user. In the *DECISION VARIABLES* section, where you define the variable vector, enter the keyword *EXPORT TO*, followed by the keyword *SPARSEFILE* and parentheses containing the filename you want **MPL** to export to.

```
Prod[i,j] EXPORT TO Sparsefile("prod.dat");
```

This will write the *activity* and the *reduced cost* values for each entry of the *Prod* variable vector to a text file in the standard sparse format. Since this file is a standard sparse data file, **MPL** can read these values back later into other models.

You can change which values will be exported by entering one or more of the following keywords directly after the keyword *EXPORT*: *Activity*, *ReducedCost*, *ObjectCoeff*, *ObjectLower*, *ObjectUpper*. If you want all of above entries you can use instead the *ALL* keyword.

```
Prod[i,j] EXPORT ALL TO Sparsefile("prod.dat");
```

MPL also allows you write solution values for variable vectors to data files in the same format as is used in the standard **MPL** solution files.

```
Prod[i,j] EXPORT TO Solutionfile("prod.sol");
```

When exporting to text files using the solution file format all the options the *Solution File Options* dialog box will have effect.

Export Variable Values to Excel Spreadsheet

MPL can also export the variable values to an Excel spreadsheet where it can be used to report the solution back to the user. Following the declaration of the variable vector, enter the keyword *EXPORT TO*, followed by the keyword *EXCEL RANGE* and parentheses containing the Excel workbook name and the Excel range name you want to export to. For example:

```
DECISION VARIABLES
  PatternCount[patterns]

  EXPORT TO EXCEL RANGE("Cutstock.xls", "PatCount");
```

In the above example, **MPL** will open the Excel spreadsheet *Cutstock.xls*, locate the range *PatCount* and then export the solution values for the variable vector *PatternCount*.

The *EXCEL RANGE* command can also be used to export activity values for plain variables. If the range given contains more than one cell, only the first cell will be used for the exported value.

You can also export the data in sparse column format where the first columns store the values for the indexes followed by a column containing the variable values. In this case, you use the *EXCEL SPARSE* command to specify that the data is to be written in a sparse format. For example:

```
DECISION VARIABLES
  ExcessCuts[cuts]

  EXPORT TO EXCEL SPARSE("Cutstock.xls", "CutsTable", 4);
```

In this example the Excel range *CutsTable* contains four columns. The first column stores the values for the index *Cuts* followed by two columns containing the widths and the demand for each cut. The activities values for the *ExcessCuts* variable vector will be stored in the fourth column of the *CutsTable* range.

If the Excel range contains the data as an Excel *List* there will be an extra row at the top of the range that contains the name of each column. **MPL** allows you to skip that row automatically by using the *EXCEL LIST* command instead of the *EXCEL SPARSE* command.

If you are writing multiple variable vectors to the same Excel spreadsheet, you can omit the workbook filename on all entries after the first one. Please note, that since variable exports are performed at a different time than imports, you will need to specify the spreadsheet workbook filename on the first export statement even if it is the same as the one you were importing from. Alternatively, you can enter the name for the Excel workbook in the *OPTIONS* section. Please refer to *Chapter 4.9* for more details on the options section.

Export Variable Values to Database

After optimizing the problem, **MPL** can export the variable values to the database where it can be used to report the solution back to the user. Following the declaration of the variable vector, enter the keyword *EXPORT TO*, followed by the keyword *DATABASE* and parentheses containing the table name and the column name you want to export to. For example:

```
DECISION VARIABLES
  FactDepShip[factory,depot]

  EXPORT TO DATABASE("FactDep","Shipment");
```

In the above example, **MPL** will open the database table *FactDep*, locate the columns *Shipment*, *FactID*, and *DepotID*, and then export the solution values for the variable vector *FactDepShip*. For further information on exporting variable values please refer to *Chapter 11.3: Export Variable Values to Database*.

8.2 Defining Macros

Macros are an important feature of the **MPL** modeling language. They can be used in several ways that can be helpful in maintaining your model, such as defining a specific part of the model into a macro that will enable you to use it repetitively, thus making the model easier to maintain. Macros can also be used to give complex expressions a distinctive name and then refer to these complex expressions throughout the model by the macro. Finally, you can use them to eliminate unnecessary constraints from the model and thereby reduce the size of the problem to be solved by the LP package.

You define each macro item with a name on the left side and a formula or an expression on the right side. Whenever the macro name is encountered in the model, the formula is used instead. The formula cannot be indexed. However, you can use summations and fixed vector items in the formula. A semicolon is required after each macro definition.

```
MACROS
  TotalRevenue := SUM(product,month: price * Sales) ;
  TotalCost    := SUM(product,month: InventoryCost * Inventory
                    + ProductionCost * Production);

MODEL

  MAX Profit = TotalRevenue - TotalCost ;
```

In this example the macros are used to give the above summations a descriptive name. The macro named *TotalRevenue* is defined as the sum of the revenues of all three products over the whole year. The *TotalCost* macro is the total cost of the inventory storage plus the total cost of the production. When these macros are encountered in the model, their respective summations they represent will be placed into the model. See *Chapter 9.9 Using Macros* for more information on how to refer to macros in the model.

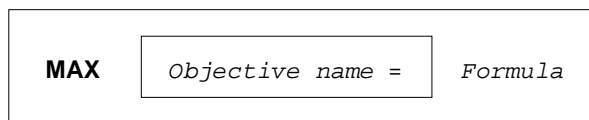
After the model has been solved, **MPL** will create a solution file that contains the solution values of the defined macros in the model. For example:

```
MACROS
```

Macro Name	Values
TotalRevenue	1298181.8182
TotalCost	753615.1818

8.3 The Objective Function

The objective function is the first part of the model formulation and is required. It starts with the keyword *MAX* or *MIN* depending on whether you want to maximize or minimize. You can also use the longer version of the keywords *MAXIMIZE* and *MINIMIZE*. The following diagram shows the basic structure of the objective function.



The name of the objective function is optional. It can be of any length, but cannot contain spaces or other delimiters. If you omit it, the name defaults to 'Z'. An equals sign follows the name. Finally, you enter the formula for the actual function.

You can extend the objective function over many lines. When it is complete, mark the end with a semicolon and/or the keyword *SUBJECT TO*.

You can use a constant value in the objective function. **MPL** places the constant directly into the generated objective function if the LP package supports that feature. If not, **MPL** will add the constant to the solution after optimizing.

Examples:

```
MAX 3x1 + 5x2 ;

MIN Labor_Cost =   3.00  Shift1
                  + 3.45  Shift2
                  + 4.50  Extra1
                  + 5.18  Extra2
                  + 120000 ;      ! fixed cost

MAXIMIZE Profit = Revenues - TotalCost ;

MINIMIZE Cost = SUM(products,months : InventoryCost) ;
```

8.4 Specifying Constraints

The constraints for the model follow immediately after the objective function. Either the keyword *SUBJECT TO* or a semicolon is required as a separator between the objective function and the constraints.

There are two types of constraints:

Plain constraints that are regular single constraints, similar to those LP packages accept in their input files.

Vector constraints that are defined over indexes and **MPL** then expands to a list constraints when generating the model.

Plain Constraints

Plain constraints are constraints that are defined without any indexes. They are normally used with constraints containing plain variables and fixed vector variables. Plain constraints can also contain summations of vector variables as long as all the indexes for the variables are accounted for. The following diagram illustrates the structure for plain constraints.

```
Constraint name : formula comparison formula ;
```

Naming the Constraint

The name of the constraint is optional. It can be of any length, but must not contain spaces or other delimiters. If you omit the name, it defaults to 'c*i*', where *i* is the number of the constraint. The constraint name must be followed by a colon.

Using meaningful names for the constraints is a good modeling practice and makes the model easier to work with. The constraint name you provide is used where applicable when **MPL** solves the model or generates input files.

The Constraint Equation

The constraint name is followed by an equation, which is entered as two formulas, separated by a comparison. The *comparison* between the formulas can be any of the following:

Less than or equal constraint: < <=
Equal constraint: =
Greater than or equal constraint: > >=

The reason there are two formulas in each constraint is so that variables can be entered on both sides of the comparison. This can enhance the readability of the model. For example if a variable has a negative coefficient it can be moved over to the other side of the comparison in order to make it positive. When **MPL** reads in the model, it moves all variables to the left side, all constants over to the right side, and changes signs where necessary.

If you refer to a variable more than once in a constraint, the coefficients will be added together in the model. For more information on writing formulas, you can refer to *Chapter 9 Building Formulas*.

Examples:

```
3 (Sb1 + Co1 + So1) = 2 (Sb2 + Co2 + So2) ;  
Overtime : Over <= 50% * 170 Workforce ;  
Production : SUM(shifts: Prod[shifts]) <= 3750 ;
```

You can extend each constraint over many lines. You must end each constraint with a semicolon ‘;’ to separate it from the next constraint. Forgetting the semicolon is one of the more common mistakes people make when formulating **MPL** models.

Vector Constraints

Very often models contain constraints that are subscripted, called vector constraints. For example, a vector constraint can be defined in this way over a number of periods and products. A vector constraint is entered by following the constraint name with a list of indexes inside brackets. The index list can contain one or more indexes. When **MPL** parses the model, it will expand each vector constraint to a list of plain constraints, based on the contents of the index list. A vector constraint is defined as shown in the diagram below:

<code>Constraint name [index list]</code> <div style="display: inline-block; border: 1px solid black; padding: 2px 10px; margin-left: 10px;"> <code>-> abbreviation</code> </div> :
--

Naming Vector Constraints

The constraint name in **MPL**, which is required for vector constraints, can be of any length. The name cannot contain any spaces or other delimiters. See *Chapter 5.2 Basic Input Elements* for more information on names and list of delimiters.

In some cases you may want to use **MPL** to generate an input file for the LP package. Most LP packages are limited to eight characters for names and therefore only the first few characters will be used when the names are built by **MPL**. Therefore, to avoid name conflicts, it is necessary to use the becomes operator ‘->’ to abbreviate the name. This feature allows you to use long, descriptive names in your **MPL** model, and at the same time meet your LP package requirements for short names.

Example:

```
SUBJECT TO
  InventoryBalance[product,month] -> Ibal:
```

Even though you can use fairly long names for constraints in **MPL** you might sometimes want to use longer names or description for documentary purposes. **MPL** allows you to do that by following the definition with the keyword *IS* and a text description. If the description contains spaces you will need to enclose it with quotation marks.

Example of the IS command:

```
SUBJECT TO
  InventoryBalance[product,month]
  IS "Balance of Inventory for each <product> per <month>";
```

Using Subranges of Indexes in Constraints

If you do not want to generate constraints for all values of a particular index you can use a *subrange* of the index. You can also use subranges when you want to use different constraints for certain values of the index, i.e. only the first index value.

Examples:

```
InventoryBalance[month=Jan..Nov] : ...
InventoryBalance[month<=Jun] : ...
InventoryBalance[month=Dec] : ...
```

The Constraint Equation

As with plain constraints, the constraint definition is followed by an equation, which is entered as two formulas, separated by a comparison. The formulas in vector constraints can contain vector variables, coefficients and summations.

Example:

```
InventoryBalance[product,period] -> InvB :
    Inventory = Inventory[period-1] + Production - Sales ;
```

In the example above, each of the variables referred to is a vector variable defined over the indexes *product* and *period*. When reading the model, **MPL** will map these variable indexes to the same indexes used to define the constraint *InventoryBalance*. Each index for a referred variable must be accounted for in either the constraint or an enclosed summation. For example, if the *Inventory* variable had been defined with a third index, that could not be mapped, **MPL** will report this with an error message.

Where Conditions on Constraints

Sometimes you want to be able to limit the number of constraints defined in a vector constraint dependent on the contents of a data vector. In this case you can use a *WHERE data condition* following right after the definition of the vector. This will result in only those constraints being defined where the condition is met.

Example of where condition:

```
SUBJECT TO
    InventoryBalance[product,month]
    WHERE (month > Jan) : ...
```

Export Constraint Values to Data File

After optimizing the problem, **MPL** can export the constraint values to separate data files which can be used to report the solution back to the user. Following the constraint declaration, enter the keyword *EXPORT TO*, followed by the keyword *SPARSEFILE* and parentheses containing the filename you want **MPL** to export to.

```
InvBal[i,j] EXPORT TO Sparsefile("invbal.dat");
```

This will write the *slack* and the *shadow price* values for each entry of the *InvBal* constraint vector to a text file in the standard sparse format. You can change which values will be exported by entering one or more of the following keywords directly after the keyword *EXPORT*: *Activity*, *Slack*, *ShadowPrice*, *RhsValue*, *RhsLower*, *RhsUpper*. If you want all of above entries you can use instead the *ALL* keyword.

MPL also allows you write solution values for constraint vectors to data files in the same format as is used in the standard **MPL** solution files.

```
ProdCap[i,j] EXPORT TO Solutionfile("prodcap.sol");
```

When exporting to text files using the solution file format all the options the *Solution File Options* dialog box will have effect.

Export Constraint Values to Excel Spreadsheet

MPL can also export the constraint values to an Excel spreadsheet where it can be used to report the solution back to the user. Following the declaration of the constraint vector, enter the keyword *EXPORT TO*, followed by the keyword *EXCEL RANGE* and parentheses containing the Excel workbook name and the Excel range name you want to export to. For example:

```
SUBJECT TO
  CutReq[cuts]:
    EXPORT ShadowPrice TO EXCEL RANGE("ShadowPrice") : ...
```

In the above example, **MPL** will open the Excel spreadsheet *Cutstock.xls*, locate the range *ShadowPrice* and then export the shadow prices for the constraint vector *CutReq*.

The *EXCEL RANGE* command can also be used to export solution values for plain constraints. If the range give contains more than one cell, only the first cell will be used for the exported value.

Part III The MPL Modeling Language

You can also export the data in sparse column format where the first columns store the values for the indexes followed by a column containing the constraint values. In this case, you use the *EXCELSPARSE* command to specify that the data is to be written in a sparse format. For example:

```
SUBJECT TO
  CutReq[cuts]:

  EXPORT ShadowPrice TO EXCELRange("CutsTable", 5) : ...
```

In this example the Excel range *CutsTable* contains four columns. The first column stores the values for the index *Cuts* followed by three columns containing the widths, the demand, and the excess for each cut. The shadow price for the for the *CutReq* constraint vector will be stored in the fifth column of the *CutsTable* range.

If the Excel range contains the data as a Excel *List* there will be an extra row at the top of the range that contains the name of each column. **MPL** allows you to skip that row automatically by using the *EXCELLIST* command instead of the *EXCELSPARSE* command.

If you are writing multiple constraint vectors to the same Excel spreadsheet, you can omit the workbook filename on all entries after the first one. Please note, that since variable and constraint exports are performed at a different time than imports, you will need to specify the spreadsheet workbook filename on the first export statement even if it is the same as the one you were importing from. Please refer to *Chapter 4.9* for more details on the options section.

Export Constraint Values to Database

If you are using the Database Connection option you can export the constraint solution values back to the database after solving the problem. This allows you to present the solution to the end-user using the reporting capabilities of the database package.

In order to export constraint solution values follow the definition of the constraint with the keyword *EXPORT TO* followed by the keyword *DATABASE* and parentheses containing the table name and the column/field name you want to export to.

```
SUBJECT TO
  FactoryCapacity[factory]
  EXPORT ShadowPrice TO DATABASE("Factory","ShadowPrice"): ...
```

In the above example, **MPL** will open the database table *Factory*, locate the columns *FactID* and *ShadowPrice*, and then export in the shadow price values for the constraint *FactoryCapacity*. For further information on exporting constraint values please refer to *Chapter 11.4: Export Constraint Values to Database*.

8.5 Bounds on Variables

Most optimizers can handle upper and lower bounds on variables efficiently without making them regular constraints. **MPL** allows you to enter bounds on variables in the *BOUNDS* section following the constraint.

```
BOUNDS
  x <= 4*12 ;
  z_bounds : 2 <= z <= 8 ;
  CloseInv : Inventory[December] = 20000;
```

Just as with constraints, you separate bounds with a semicolon. To make your formulation easier to read, you can use bound names in the same way as constraints. The bound names can be of any length, but cannot contain spaces or other delimiters.

You can enter variable bounds in any of the following forms:

- 1) variable >= constant ; { lower bound }
- 2) variable <= constant ; { upper bound }
- 3) constant <= variable ; { lower bound }
- 4) constant >= variable ; { upper bound }
- 5) variable = constant ; { fixed variable }
- 6) constant = variable ; { fixed variable }
- 7) constant <= variable <= constant;
- 8) constant >= variable >= constant;

The first six are the same input forms you would use for standard constraints. Forms 5 and 6 are used to fix a variable to a constant value so it will not be changed during the optimization. Forms 7 and 8 can be used when a variable has both lower and upper bounds to make the expression simpler and easier to read.

The constant values in bounds can be entered using standard coefficient arithmetic. For more information, refer to *Chapter 9.1: Coefficients for Variables*.

Bounds on Vector Variables

Bounds on vector variables can be entered with a bound name and a corresponding index list for the bound, similar to the way vector constraints are specified.

You can also enter the bound without a bound name and the index list. In this instance **MPL** will look up the declaration of the variable for the index list.

Part III The MPL Modeling Language

If you want to use a subrange of an index, instead of the declared index you can enter it inside the brackets.

```
BOUNDS
  MaxInv[month<=Nov] : Inventory <= 90000 ;
  Inventory[month=Dec] = 90000 ;
  Sales <= Demand ;
```

Free Variables

MPL assumes that all decision variables are non-negative. You can override that assumption, by defining the variables as “free”, which will then make their values unrestricted. To define free variables, use the keyword *FREE* followed by a list of the variables that are free. This free section can be placed anywhere after the constraints section. The variables can be both plain and vector variables.

```
FREE
  Inventory[month] ;
```

In this example all the inventory variables will be declared as free variables.

Semi-Continuous Variables

MPL also allows you to define semi-continuous variables for solvers that support that feature. You specify a variable to be semi-continuous by using the keyword *SEMICON* followed by the variable name. The semi-continuous section can be placed anywhere after the constraint section.

```
BOUNDS
  MinMach <= MachineHours[machine] <= MaxMach;

SEMICON
  MachineHours[machine] ;
```

In this example, the *MachineHours* variable is defined as semi-continuous with a feasible range which includes any value between its upper and lower bounds as well as zero.

8.6 Integer and Binary Variables

MPL can handle both integer and binary variables. To define integer variables, use the keyword *INTEGER* followed by list of the variables that are integer. The integer section can be placed anywhere after the constraints section. The variables can be both plain and vector variables. For more information on decision variables see *Chapter 8.1 Declaring Decision Variables*.

Binary variables are integer variables that can only have values of zero or one. You define them in the same way as integer variables, using the keyword *BINARY* instead.

```
INTEGER
  Production[month,product] ;
BINARY
  ShopOpen ;
```

MPL will direct the solver to use integer and binary variables if they are supported by the solver.

Special Ordered Sets of Variables

MPL allows you to define Special Ordered Sets (SOS) of variables in the model. For each set, use one of the keywords *SOS1*, *SOS2*, or *SOS3*, and follow it with a list of the variables that belong to the set. This section can be anywhere after the constraints section. The variables can be either plain or structured. If you need to specify the reference row use the keyword *REFERENCE* immediately after the *SOS* keyword.

```
SOS1
  x1, x2, x3 ;

SOS2 REFERENCE Capacity;
  Produce[product];
```

Please check if the solver you are using supports SOS by referencing the solver documentation. **MPL** will generate the sets for the solver if they are supported, but will otherwise ignore them.

MPL also supports specifying multiple SOS sets for subsets of a vector variable range. This can be very useful when you have a vector variable with multiple indexes, but there is a separate set for each element of one or more of the indexes. The indexes that are used in each set are specified using the *SET* keyword as shown here below:

```
SOS1
  s[i]: SET(k: x[i,k]);
```

This will create a separate SOS set for each *i* with each set containing the *x* variables for all the elements of the *k* index.

Part III The MPL Modeling Language



CHAPTER 9

BUILDING FORMULAS

9.1 Coefficients for Variables

Every variable has a coefficient in each constraint, which you can specify. If the variable is not mentioned in the constraint the coefficient value is zero. If there is a variable, but no coefficient is given, the default value is one. The coefficient can either be placed in front of the variable or trailing it.

You can enter coefficients as simple numbers or as any combination of products, fractions, percentages, and arithmetic functions. Parentheses containing constant expressions can also be used in the denominators of the coefficients.

MPL evaluates those expressions when it reads the model file, and puts the results in the input file. This capability enables you, for example, to enter fractions with complete accuracy. Named data constants declared in the *DATA* section can be used as you would any other numbers.

Examples of coefficients:

4	{	= 4.0	}
1/3	{	= 0.3333333333	}
1/(3+5)	{	= 0.125	}
2 * InvtCost	{	= 2 * 8.8 = 17.6	}
1.32 / 24.5% * 53	{	= 285.55102041	}

In the fourth line it is assumed that the *InvtCost* is a named data constant which has been defined as 8.8.

9.2 Using Arithmetic Functions

Arithmetic functions are available in **MPL** when specifying coefficient values. They can be used in the model wherever you use coefficients.

The available functions are listed here below:

Arithmetic functions:

ABS(x)	Absolute value of x
EXP(x)	Exponential of x
LOG(x)	Natural logarithm of x
LOG10(x)	10 logarithm of x
PI()	The value of Pi
POWER(x,n)	Raises a number x to a power of n
RANDOM(n)	Random generated value
SIGN(x)	Sign of x
SQR(x)	Square of x
SQRT(x)	Square root of x

Trigonometric functions:

ACOS(x)	Inverse cosine of x, $\cos^{-1}(x)$
ACOSH(x)	Inverse hyperbolic cosine of x, $\cosh^{-1}(x)$
ASIN(x)	Inverse sine of x, $\sin^{-1}(x)$
ASINH(x)	Inverse hyperbolic sine of x, $\sinh^{-1}(x)$
ATAN(x)	Inverse tangent of x, $\tan^{-1}(x)$
ATANH(x)	Inverse hyperbolic tangent, $\tanh^{-1}(x)$
COS(x)	Cosine of x, $\cos(x)$
COSH	Hyperbolic cosine of x, $\cosh(x)$
SIN(x)	Sine of x, $\sin(x)$
SINH	Hyperbolic sine of x, $\sinh(x)$
TAN(x)	Tangent of x, $\tan(x)$
TANH(x)	Hyperbolic tangent, $\tanh(x)$

Integer functions:

CEIL(x)	Next higher integer value of x
FLOOR(x)	Next lower integer value of x
ROUND(x)	Rounding to the nearest integer value of x
TRUNC(x)	Truncating to the nearest integer value of x

Data vector functions:

MAX(A)	Maximum value of a data vector A
MIN(A)	Minimum value of a data vector A
AVG(A)	Average value of a data vector A

The arithmetic functions expect a constant value (which can be a coefficient formula) or a data vector as argument, and return a real number. The functions operate on real numbers, except *SQRT*, *LOG*, and *LOG10* that take only positive numbers, *EXP* that takes only values less than 40, *PI* that has no arguments, and *RANDOM* that takes either no argument (returns real number between 0 and 1) or whole numbers (returns a whole number less than argument). You can also set a new seed value for the *RANDOM* function by adding *seed=number* after the argument. The integer functions expect a constant value as argument and return a whole number which can be used in subscript formulas. The data vector functions expect a defined data vector as argument and return a real number.

Examples of functions:

```
ATAN(3.1415);  
SQR(183/2);  
RANDOM(100,seed=42);  
AVG(Demand);
```

9.3 Using Variables in Formulas

The decision variables are the elements under control of the model developer and their values determine the solution of the model. Decision variables in **MPL** are defined as they come in the input file. The name can be of any length. It starts with a letter, with the remainder consisting of letters and digits. **MPL** is case sensitive by default so you need to be careful to distinguish between upper and lower case letters. If you want **MPL** not to be case sensitive you can change the default by selecting *MPL Language* in the *Options* menu. See *Chapter 4.9: The Options Menu* for further details.

There are two types of decision variables: plain variables and vector variables (some-times called subscripted variables). A plain variable is setup as a single column in the matrix that is sent to the LP solver. A vector variable, on the other hand, is setup as range of columns in the matrix. The *DECISION VARIABLES* section is used to define the vector variables. See *Chapter 8.1: Declaring Decision Variables* for further details.

The most important thing to remember when referring to vector variables is that all the indexes defined for the vector must be accounted for in the domain, either by the constraint specification, the enclosing summation, or the subscript.

Vector variables can be referred in the model in a number of ways. The quickest is to enter the vector without the indexes and let **MPL** figure it out from the definition. Another way is to write the vector as it was defined, with all the indexes listed, for example *Inventory[product,month]*. The order of the indexes is actually irrelevant; **MPL** gets the correct order from the definition.

Sometimes the index of the constraint and the index of the vector may have to differ by some offset, for example, when the inventory for each month depends on the inventory of the previous month. You can enter this by using an *offset* value, which can be specified as a constant, a data vector or an index. To specify the offset value, the appropriate index is followed by either a plus or minus sign and the slide value. For example:

```
Inventory = Inventory[month-1] + Production - Sales;
```

When generated, the last constraint would look like this:

```
Inv1Dec = Inv1Nov + Prod1Dec - Sale1Dec;
```

In the constraint for the month of December, the November subscript is used for the initial inventory. This method works for any integer offset value, and the relevant index is the only one that must be specified. Entries that fall out of bounds are ignored. For example, the constraint for January would be:

```
Inv1Jan = Prod1Jan - Sales1Jan;
```

Values of index variables can also be fixed or set to a specific value, rather than range over the index domain. Simply use the value in place of the index variable. In that case, the order of the indexes becomes important. As a rule, when a fixed value is used, all indexes should be specified in the same order as when the vector was defined. For example:

```
Inventory[1,December] = 20000;
```

Another way to implement fixed values is to use subindexes in the constraint specification with a range of a single value. For more details see *Chapter 8.4: Specifying Constraints*.

9.4 Formula Terms

The building block of the objective function and the constraints is a formula which specifies the coefficients to the variables. Variables that are not mentioned will automatically have a coefficient of zero.

When you are working with the objective function or a simple constraint, the formula term consists of a variable and its associated coefficient, or just a single constant. The coefficient can either be placed in front of the variable or trailing it. These terms are then added together to make a formula.

Examples of formulas:

```
200 Shift1ProdA + 400 Shift1ProdB
50% * 170 Workforce - HoursOvertime
3x + y/3 - 67.33*47% z + 1.32/(45%+10%)
```

Terms in structured constraints consist of coefficients, up to eight data vectors, and a decision vector. For example, the following terms for the vector variable *Inventory* are legal:

```
8.8 Inventory
InvCost * Inventory
12%/2 * InvCost * InvCount * Inventory
```

Note that the multiplication performed is a element wise multiplication, not a matrix multiplication. When the vectors being multiplied have the same set of index variables, the result is what you would expect. However, when the index variable sets are not identical, the vectors are expanded and replicated in the missing dimensions.

Chapter 9 Building Formulas

For example, if *Cost* and *Sales* are defined by:

```
DATA
  Cost[product] = (1,2,3);
  Sales[month]  = (12,16,18,20);
```

the vectors are first expanded to:

```
Cost = (1,2,3)   Sales = (12,12,12)
       (1,2,3)   (16,16,16)
       (1,2,3)   (18,18,18)
       (1,2,3)   (20,20,20)
```

and then multiplied element by element to obtain:

```
Cost * Sales = (12, 24, 36)
               (16, 32, 48)
               (18, 36, 54)
               (20, 40, 60)
```

When two or more variables share the same coefficient i.e. the coefficients have a common divisor, you can factor them out using parentheses. This will be explained better in the *Parentheses* section later in this chapter.

Simple terms are valid in structured constraints and get expanded appropriately. Sums and macros are also valid terms and are discussed in later sections in this chapter.

9.5 IF/IIF Conditions on Formula Terms

In some cases, a term is not included in a formula or you need to choose between two terms based on some data values. In those instances, you can use either an *IF THEN ELSE* condition, or *IIF* (Immediate IF) function to specify when each term should be included. For example, if you only want to include variable vector *Produce*, if the corresponding data vector value for cost is greater than zero, you can enter this as follows:

```
... IF (Cost[product] > 0) THEN Produce[product] ENDIF + ...
```

The same term can be entered using the *IIF* function statement which sometimes is more readable:

```
... IIF(Cost[product] > 0, Produce[product]) + ...
```

When you are choosing between two terms based on a data value, you can use the *ELSE* statement to specify the alternative term. For example:

```
InvtBal[prod, month]:  
  IF (month = 1) THEN StartInvt[prod] ELSE Invt[month-1] ENDIF + ...
```

Or, alternatively, using the *IIF* function statement:

```
InvtBal[prod, month]:  
  IIF(month = 1, StartInvt[prod], Invt[month-1]) + ...
```

9.6 Where Conditions on Formula Terms

Sometimes it is necessary to make a term formula, dependent on some data values. **MPL** offers a feature called the *WHERE condition*, which means that certain terms are included only where a condition on the data is met. The *WHERE* command is similar to the *IF/IIF* statements and is often more readable, especially in cases where there is no *ELSE* term.

For example, assume that you have a data vector $A[i,j]$ filled with sparse data and you only want the following constraint where the values are greater than zero. You would write:

```
constr[i,j] WHERE (A[i,j] > 0.0) : ...
```

The same rule applies when you want to use the *WHERE* condition on formula terms:

```
+ x[i] EXCEPT WHERE (b[i] = 1.0) + ...
```

In this example the variable $x[i]$ is used wherever $b[i]$ is not equal to one. The *EXCEPT* keyword inverts the set of values.

9.7 Referring to Indexes in Formulas

Indexes can be used in formulas as any other data element. As in previous versions, all indexes must be accounted for in the underlying index list (sum or constraint specification). Here are two examples to illustrate this:

```
DATA
  A[i,j] := DENSE(10*(i-1)+j);
MAX z = SUM(i: i*x[i]);
```

Furthermore, the integer functions *FIRST(index)*, *LAST(index)*, and *COUNT(index)* are available. They provide the first subscript, the last subscript, and the number of subscripts, respectively, of the given index and can be used both in formulas and in subscript arithmetic.

```
INDEX
  i := 1..10;
  j := 1..last(i);
  k[i] := (first(i), last(i));

DATA
  A[i] := i;
  B[i] := DENSE(count(i)-i+1);
```

9.8 Using Parentheses

When two or more variables share the same coefficient, i.e. the coefficients have a common divisor, you can factor them out using parentheses. With parentheses the model formulation becomes both simpler and easier to work with.

The parentheses contain decision variables and even full-fledged formulas. Those formulas can, in turn, have more parenthesized expressions, so nested parentheses are allowed. You can enter coefficients in front of the parentheses.

Examples of parentheses:

$1/30 (\text{PrA} + 2 \text{PrB})$	{ = 0.0333 PrA + 0.0666 PrB }
$5 (x + 10 (y + 2z))$	{ = 5x + 50y + 100z }
$1/(3 + 7) x$	{ = 0.1 x }

Four main rules apply when you use parentheses. Parenthesized expressions cannot be:

1. Multiplied together.
2. Multiplied by a variable.
3. Used in a denominator if they contain variables.
4. Followed by a coefficient.

These rules ensure that the problem formulation stays linear.

Examples of illegal parentheses:

$(15 + 200)(5x + 3y)$	{ Rule 1 }
$(15 + 200) x1$	{ Rule 2 }
$1 / (x + 200) \text{Prod1}$	{ Rule 3 }
$(x + y) / 3$	{ Rule 4 }

9.9 Summing Vectors Over Index Values

Summations are used when you want to sum vector terms over some indexes. This consists of using the keyword *SUM*, the parentheses containing the list of indexes summed, and the summation formula.

Both variable and data vectors can be included in the summation formula. The vectors can be added and subtracted, and parentheses and other summations are allowed within the sum. This example shows how sums are used:

```
MaxProd : SUM(month : Production) < 1200;
```

The above constraint means that sum of the *Production* variables for all months has to be less than 1200. The resulting constraint in the input file generated, is:

```
MaxProd : ProdJan + ProdFeb + ProdMar + ProdApr
          + ProdMay + ProdJun + ProdJul + ProdAug
          + ProdSep + ProdOct + ProdNov + ProdDec < 1200;
```

Now assume that you have three products and want to limit the production for each. Then the constraint would be:

```
MaxProd[product] : SUM(month: Production) < ProdCapacity;
```

This summation generates three constraints, one for each product. The *Production* vector is now defined over the indexes *product* and *month*. The constraint for a given product contains the sum of the *Production* variables for all months, but with the product fixed.

The main rule when using sums is that all declared indexes of a vector term must be accounted for, either by the sum indexes, the constraint indexes, or by a fixed subscript.

When summing over a single vector, you can omit the list of index variables. **MPL** automatically uses the dimension of the vector when it expands the sum.

9.10 Using Macros

Macros can be used in several ways that can be helpful in maintaining your model; such as defining a specific part of the model into a macro that will enable you to use it repetitively, thus making the model easier to maintain. Macros can also be used to give complex expressions a distinctive name and then refer to these complex expressions throughout the model by the macro. Finally, you can use them to eliminate unnecessary constraints from the model and thereby reduce the size of the problem to be solved by the LP package.

A defined macro can be used in the formula of a constraint as a term. The macro is expanded as it is replaced by the expression it represents. Macro terms can be preceded by coefficients.

Example:

```
MACROS
  Revenues := SUM(product,month: price * Sales) ;
  TotalCost := SUM(product,month: InventoryCost * Inventory
                    + ProductionCost * Production);

MODEL

  MAX Profit = Revenues - TotalCost;
```

Macros are helpful to aid readability by giving complex expressions a distinctive name. They are also used to eliminate unnecessary constraints from the model and thereby reduce the size of the problem to be solved by the LP package. See *Chapter 8.2 Defining Macros* for more information.

9.11 Abort If Conditions

The *ABORT IF* conditions are used when you want to make sure that a certain item in the **MPL** model is defined correctly. They can be a very important tool to validate your model especially when importing indexes and data from external sources.

In the example below, **MPL** will stop and display an error message because the index *Top3Plants* does not contain 3 elements.

```
INDEX
  plants := (Atlanta, Chicago, Dallas, NewYork, SanDiego);
  Top3Plants[plants] := (Atlanta, Chicago, Dallas, SanDiego);
ABORT IF NOT (count(Top3Plants) = 3)
  MESSAGE "The number of plants in Top3Plants must be 3";
```

In the example below, we are checking if two subset indexes, which should be mutually exclusive, have any common elements. As both indexes contain the city *Dallas*, **MPL** will stop and issue the error message.

```
INDEX
  EastCoast[plants] := (Atlanta, Chicago, Dallas, NewYork);
  WestCoast[plants] := (Dallas, SanDiego);
ABORT IF FORSOME(plants IN EastCoast: plants IN WestCoast)
  MESSAGE "One of the EastCoast plants is also in WestCoast";
```

You can also use the *ABORT IF* condition to check if the values of the data vectors are within the allowed range. For example, in the example below, the capacity for each plant must be at least 200.

```
DATA
  Capacity[plants] := (120, 300, 250, 400, 500);
ABORT IF FORSOME(plants: Capacity[plants] < 200)
  MESSAGE "Capacity for each plant must be at least 200);
```

Part III The MPL Modeling Language



CHAPTER 10

ADVANCED INDEXING

TECHNIQUES

10.1 Set Membership with the IN Operator

The *IN* operator in **MPL** allows you to select one of the domain indexes from a multidimensional index. For example, if you have a multidimensional index that specifies which machines are available in which plants, you can use the *IN* operator to sum over all the machines in each plant.

```
INDEX
  plant      := (Atlanta, Chicago, Dallas);
  machine    := (Grind, Drill, Press);
  product    := (p1,p2,p3,p4);

  PlantMach[plant,machine] := (Atlanta.Grind, Atlanta.Press,
                               Chicago.Drill, Chicago.Press,
                               Dallas.Grind);
  .
  .
SUBJECT TO
  PlantCapacity[plant] :

    SUM(machine IN PlantMach: Prod[machine]) < MaxCapacity[plant];
```

In the above example, we sum together how much is produced with each machine and then make sure that the total production is limited to the maximum capacity at each plant.

We can also use the *IN* operator to join together two multi-dimensional indexes. For example, say that you also have an index that defines which products are produced by which machine. You can then use the *IN* operator, in conjunction with the *WHERE* condition, to create a new index that contains which products can be produced in which plants.

```
INDEX
  MachProd[machine,product] := (Grind.p1, Grind.p4,
                               Drill.p2, Drill.p3, Drill.p4,
                               Press.p1, Press.p3);

  PlantProduct[plant,product] WHERE

    FORSOME(machine: machine IN PlantMach = machine IN MachProd);

  !
  !           := (Atlanta.p1, Atlanta.p3, Atlanta.p4,
  !             Chicago.p1, Chicago.p2, Chicago.p3, Chicago.p4,
  !             Dallas.p1, Dallas.p4);
```

In the example on the previous page, we join the plants and the products together through the *machine* index. This is done by **MPL** going through each *plant.machine* pair in the *PlantMach* index and matching the machine to the *machine.product* pair in the *MachProd* index. This will give us a new index *PlantProduct* which contains, for each plant, the products that can be produced at that plant. If you are familiar with relational databases and view the indexes in **MPL** as columns, then you will notice that this operation is the same as the *natural join* operation in databases.

Chapter 10 Advanced Indexing Techniques

When formulating the model with a lot of sparsity, you can use the *IN* operator to let **MPL** quickly skip index elements that are not needed. For example, say you want to define a produce variable for each plant, machine, and product, but you only want to include those variables that are members of the two parent indexes *PlantMach* and *MachProd*. This can be accomplished in **MPL** by using the *IN* operator when you are listing the domain indexes for the variable. For example:

```
DECISION VARIABLES
  Produce[plant, machine IN PlantMach, product IN MachProd];
```

MPL will now include a *machine* index element only in the *Produce* variable if the *plant.machine* pair is in the *PlantMach* index and then a *product* index element only if the *machine.product* index is in the *MachProd* index. This will result in the total number of variables defined to be much smaller and speed up the parsing time for the rest of the model considerably, especially for large, sparse models.

10.2 Set Domain Index with the Dot Operator

Sometimes, when working with multi-dimensional indexes, you need to be able to refer to the underlying domain index. This can be accomplished in **MPL** by using the *dot* operator. Here is an example:

```
INDEX
  FromNode := (n1,n2,n3,n4);
  ToNode   := FromNode;

  Arcs[FromNode,ToNode] := n1, n2, n2, n2, n2, n3, n3, n4);
  Ards2[FromNode, ToNode] := Arcs WHERE (FromNode <> ToNode);

DECISION VARIABLES
  Ship[Arcs] WHERE (Arcs.FromNode <> Arcs.ToNode);

END
```

In the above example, we want the *Ship* variable to contain all the elements of the index *Arcs*, where the *FromNode* is not the same as the *ToNode*.

10.3 Set subsets with the OVER operator

The *OVER* operator can be used in summations and other underlying index lists. This is useful, for example, when you need to sum over a subset of a multi-dimensional index that contains certain domain index element. Here is an example:

```
INDEX
  depot := ...
  factory := ...
  FactDepot[factory,depot] := ...

SUBJECT TO

  DepotCapacity[depot]:

    SUM(FactDepot OVER factory: Ship[FactDepot]) <= DepotCap[depot];
```

In this example we need to sum over the factories that ship to the depot and make sure that the total amount shipped is less than the capacity for the depot. Notice that in this example the *Ship* variable is defined with the *FactDepot* index.

10.4 Subscript Arithmetic with Conditional Indexes

Sometimes, you do not want to use every element of a given index when expanding vectors. In **MPL** you can accomplish this by using *conditional*. This feature allows you to set conditions on the indexes in the index list that are based on current values of the other indexes. The following example explains this better:

```
SUM(i,j<i: formula)
```

The above condition defines a double sum over the indexes i and j with the additional condition that it only sums the values where j is less than i , i.e., the lower triangular matrix, excluding the diagonal. The conditions can also contain more complex subscript arithmetic. For example:

```
constr[i,k=2i+3] : ...
```

This example generates a constraint for all i and k where k is equal to $2i+3$. The other constraints are still generated, but they do not contain any terms and are therefore specified as empty constraints. **MPL** will not send the empty constraints to the solver.

Please note that conditional indexes are also allowed in the definition part when you are defining data and variable vectors. However, these conditions do not have any effect until the vectors are used in the model formulation. For example, when you define the data vector $A[i,j<=i]$ in the *DATA* section, the list of numbers that follows must also contain values for the elements where j is greater than i . The condition is not used until the data vector is referred to later in the model.

You can put a condition on every index in vectors, summations, and constraints. Be sure to keep in mind that this powerful tool can easily make the model very complex and, therefore, harder to maintain.

10.5 Direct Assignment of Subscripts

Sometimes, when formulating models, the use of indexes is too complicated for any of the previously mentioned indexing techniques. In those instances, you can, in most cases, use the *direct assignment* of subscripts. This allows you to give a referred index whatever subscript value you need, instead of having to work from the subsets of the original index. Direct assignments are entered, following the index, with an assignment symbol ($:=$) and an index formula.

To give you an example, when formulating transshipment problems, you frequently encounter constraints similar to this one:

```
Continuity[k] :
    Prod[k] + SUM(i: Ship[i,j:=k]) = SUM(j: Ship[i:=k,j]) + Sales[k];
```

In the above example, we want to sum on the left-hand side, how much is shipped to location k , and on the right-hand side, how much is shipped from location k . Since the *Ship* variable is defined over the indexes i and j , we use direct assignment to assign them the value of k .

Sometimes you need a full-fledged formula to specify the subscripts for the index.

```
SUM(i: x[i,j:=last(i)-i+1])
```

In this example, direct assignment is used to let the subscript j decrease as the subscript i increases.

Assignment of subscripts can also be used to perform multiplication of matrixes and vectors as shown here below:

```
INDEX
  i := 1..3;
  i2 := i;
  j := 1..4;

DATA
  A[i,j] := (1, 2, 3, 4,
             5, 6, 7, 8,
             9,10,11,12);

! Transpose of matrix A
  B[j,i] := A[i,j];

! Perform matrix multiplication
  AmultB[i,i2] := SUM(j: A[i,j] * B[j,i:=i2]);
```

Part III The MPL Modeling Language



CHAPTER 11

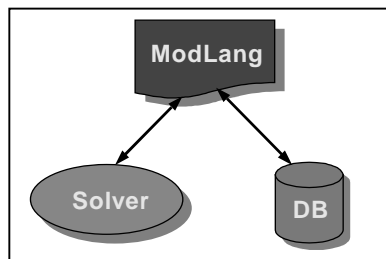
DATABASE CONNECTION

Importing data from a variety of corporate database systems into optimization models is frequently an essential requirement for optimization projects. One of the advanced features of **MPL** is the database connection option that directly links **MPL** with relational databases and other data sources. This option enables the model developer to gather both indexes and data values from various data sources and import them directly into the model. After the model has been optimized, the solution output can be exported back into the database. This, along with the run-time features of **MPL**, allows the model developer to easily create customized end-user applications for optimization using the built-in data entry and reporting capabilities of the database.

The database connection in **MPL** has the ability to access data from many different sources, such as databases, Excel spreadsheets, external data files, and the Internet. This gives the model developer the flexibility to choose the most efficient and convenient way to incorporate the data into the model. Among the data formats that are supported by **MPL** are: *Microsoft Access* and *Excel*, *ODBC*, *Paradox*, *FoxPro*, *DBase*, *SQL Server*, and *Oracle*.

Part III The MPL Modeling Language

Relational databases are designed to store and retrieve structured data enabling the user to access corporate data such as plants, products, machines, and etc., on the symbolic level. The purpose of the modeling language is to take this structured data and generate a mathematical matrix from the model that the optimization solver can process.



One alternative to using a modeling language is to write a customized matrix generator in a programming language. This kind of programming is very difficult and causes the resulting application to be highly dependent on specific methods and libraries, of both the database and the solver. **MPL** offers clear benefits to this approach, because with the database connection it can automatically map the columns of the database tables to the indexes and data vectors of the model without involving any programming. This gives the model developer unprecedented flexibility and expressive power and the ability to focus mainly on formulating the model and connecting the data instead of programming.

In the following sections, we will describe each import and export operation. Refer to the *Database Options* dialog box in *Chapter 4.9: The Options Menu* to select the database you will be using.

11.1 Import Indexes from Database

MPL allows you to import the elements for an index directly from a database. In the *INDEX* section, where you define the index, enter the keyword *DATABASE* after the assignment symbol (*:=*) followed by parentheses containing the table name and the column/field name you want to import from.

```
INDEX
  depot := DATABASE("Depots","DepotID");
```

In the above example, **MPL** will open the database table *Depots*, locate the column *DepotID*, and then read in the entries for the index *depot*. In most cases the imported indexes are the key fields for the table which are underlined in the following examples.

The Depots Table:

<u>DepotID</u>	Capacity
Atlanta	400000
Chicago	50000
NewYork	70000
Dallas	100000

The column name defaults to the name of the index so if it is the same you do not have to specify it. In the example below the column name in the database table is *DepotID* which is the same as the index *DepotID*.

```
INDEX
  DepotID := DATABASE("Depots");
```

MPL supports multiple databases. The default database is specified in the *Database Options* dialog box in the *Options* menu. See *Chapter 4.9 The Options Menu* for more information.

Part III The MPL Modeling Language

MPL can import from more than one database in the same run. If you need to import an index from table in a database other than the default, you can do so by specifying the database name before the name of the table. In the example below, **MPL** will read in the *factory* index from **Access**, instead of the default database.

```
INDEX
  factory := DATABASE(Access,"Factory","FactID");
```

MPL also allows you to import subset indexes from database tables.

```
INDEX
  FactoryDepot[factory,depot] := DATABASE("FactDep");
```

The above statement will open the database table *FactDep* and locate the columns for *factory* and *depot* and then read in the entries for *FactoryDepot*.

The FactDep Table:

<i>FactID</i>	<i>DepotID</i>	<i>TrCost</i>	<i>Shipment</i>
Houston	Chicago	3200	0
Houston	Dallas	5100	0
Seattle	Atlanta	2800	0
Seattle	Chicago	6800	0
Seattle	NewYork	4700	0
Seattle	Dallas	5400	0

Notice that **MPL** automatically uses the same name as default for the columns *FactID* and *DepotID*, as in the original tables the indexes were defined from. If an index column does have a different name than in the original table, you can specify it following the table name by first entering the index name followed by an equal sign and the column name.

```
INDEX
  FactoryDepot[factory,depot] :=
    DATABASE("FactDep",factory="Factory",depot="Depot");
```

This means if you are consistent in naming the columns in different tables you do not have to specify them each time you refer to them in **MPL**.

11.2 Import Data Vectors from Database

MPL allows you import the elements for a data vector directly from a database. In the *DATA* section, where you define the data vector, enter the keyword *DATABASE* after the assignment symbol (*:=*), followed by parentheses containing the table name and the column/field name you want to import from.

Example:

```
DATA
FactDepCost[factory,depot] := DATABASE("FactDep","TrCost");
```

In the above example, **MPL** will open the database table *FactDep*, locate the columns *TRCost*, *FactID*, and *DepotID*, and then read in the entries for the data vector *FactDepCost*.

The FactDep Table:

<i>FactID</i>	<i>DepotID</i>	<i>TrCost</i>	<i>Shipment</i>
Houston	Chicago	3200	0
Houston	Dallas	5100	0
Seattle	Atlanta	2800	0
Seattle	Chicago	6800	0
Seattle	NewYork	4700	0
Seattle	Dallas	5400	0

Notice that **MPL** automatically uses the same name as the default for the index columns *FactID* and *DepotID* as in the original tables that the indexes were defined from. If an index column does have a different name than in the original table, you can specify it following the table name by first entering the index name, followed by an equal sign and the column name.

```
DATA
FactDepCost[factory,depot] :=
DATABASE("FactDep","TrCost",factory="Factory",depot="Depot");
```

Part III The MPL Modeling Language

This means if you are consistent in naming the columns in different tables you do not have to specify them each time you refer to them in **MPL**.

The column name defaults to the name of the data vector, so if it is the same you do not have to specify it. In the example below the column name in the database table is *TrCost* which is the same as the data vector *TrCost*.

```
DATA
  TrCost := DATABASE("FactDep");
```

In some instances you do not want to read all the data elements that are in the table. In that case, you can enter the keyword *WHERE* followed by a condition on one of the columns. Here is an example:

```
DATA
  FactDepCost[factory,depot] :=
  DATABASE("FactDep","TrCost" WHERE Region="NorthWest");
```

In this example we only want to read the transportation costs from the *FactDep* table where the *Region* column contains the entry *NorthWest*.

MPL can import from more than one database in the same model. The default database is specified in the *Database Options* dialog box in the *Options* menu. If you need to import a data vector from table in a database other than the default, you can do so by specifying the database name before the name of the table. Alternatively, you can use the *OPTIONS* keyword to specify another database to import from.

In the example below, **MPL** will read in the data vector *DepotCustCost* from **Access** instead of the default database.

```
DATA
  DepotCustCost := DATABASE(Access,"DepCust","TrCost");
```

11.3 Export Variable Values to Database

After optimizing the problem, **MPL** can export the variable values to the database where it can be used to report the solution back to the user. In the *DECISION VARIABLES* section, where you define the variable vector, enter the keyword *EXPORT TO* after the defined variable, followed by the keyword *DATABASE* and parentheses containing the table name and the column/field name you want to export to.

Example:

```
DECISION VARIABLES
FactDepShip[factory,depot]

EXPORT TO DATABASE("FactDep","Shipment");
```

In the above example, **MPL** will open the database table *FactDep*, locate the columns *Shipment*, *FactID*, and *DepotID*, and then export the solution values for the variable vector *FactDepShip*.

Here is an example of the *FactDep* table after the solution values have been exported.

The FactDep Table:

<i>FactID</i>	<i>DepotID</i>	<i>TrCost</i>	<i>Shipment</i>
Houston	Chicago	3200	210
Houston	Dallas	5100	0
Seattle	Atlanta	2800	455
Seattle	Chicago	6800	0
Seattle	NewYork	4700	328
Seattle	Dallas	5400	189

Notice that **MPL** automatically uses the same name as the default for the index columns *FactID* and *DepotID*, as in the original tables the indexes were defined from.

Part III The MPL Modeling Language

If an index column does have a different name than in the original table you can specify it following the table name by first entering the index name followed by an equal sign and the column name.

```
DECISION VARIABLES
FactDepShip[factory,depot]
EXPORT TO DATABASE("FactDep", "Shipment"
                  factory="Factory",
                  depot="Depot");
```

This means, if you are consistent in naming the columns in different tables, you do not have to specify them each time you refer to them in **MPL**.

The column name defaults to the name of the data vector so if it is the same you do not have to specify it. In the example below, the column name in the database table is *FactDepShip* which is the same as the data vector *FactDepShip*.

```
DECISION VARIABLES
FactDepShip[factory,depot]
EXPORT TO DATABASE("FactDep");
```

MPL supports multiple databases for both importing and exporting data. The default database is specified in the *Database Options* dialog box in the *Options* menu. If you need to export a variable vector to table in a database other than the default, you can do so by specifying the database name before the name of the table.

In the example below, **MPL** will export the solution values of the variable vector *FactDepShip* from **Paradox** instead of the default database.

```
DECISION VARIABLES
FactDepShip[factory,depot]
EXPORT TO DATABASE(Paradox, "FactDep", "Shipment");
```

MPL also allows you to export variable values other than the activity. You can export the reduced costs, the upper and lower ranges for the objective function, as well as the objective function coefficient values. You can change which values will be exported by entering one of the following keywords directly after the keyword *EXPORT*: *Activity*, *ReducedCost*, *ObjectCoeff*, *ObjectLower*, *ObjectUpper*. For example, if you want to export the reduced cost for a variable enter the following:

```
DECISION VARIABLES
FactDepShip[factory,depot]
EXPORT ReducedCost TO DATABASE("FactDep","ReducedCost");
```

If you need to export more than one value for a variable vector you can do so by entering multiple export statements after the variable definition.

MPL offers three different options on how the database table is updated by the *EXPORT* statement. The first one, which is the default, searches through the database and for each record locates the corresponding value in the solver solution and updates the database entry with it. This option minimizes the changes done to the database table since only the existing values are updated, but can sometimes be slow especially on SQL type databases.

The second option is to use the *REFILL* keyword right after the *EXPORT* keyword to specify that the whole database table should be emptied and then refilled with the entries from the solver solution. Since this takes out the necessity to search the table this can often lead to faster export times for larger tables.

The third option is to use the *CREATE* keyword right after the *EXPORT* keyword to specify that the database table should be created and then filled with the entries from the solver solution. This option is mainly useful when exporting to the database table for the first time.

11.4 Export Constraint Values to Database

Just as exporting variable values, **MPL** can also export the constraint values to the database. You will need to define constraint vectors in the *CONSTRAINTS* section with the keyword *EXPORT TO* followed by the keyword *DATABASE* and parentheses containing the table name and the column/field name you want to export to.

Example:

```
SUBJECT TO
  FactoryCapacity[factory]
  EXPORT ShadowPrice TO DATABASE("Factory","ShadowPrice") :
```

In the above example, **MPL** will open the database table *Factory*, locate the columns *FactID* and *ShadowPrice*, and then export in the shadow price values for the constraint *FactoryCapacity*.

Here is an example of the *Factory* table after the shadow price values have been exported.

The Factory Table:

<i>FactID</i>	<i>Capacity</i>	<i>ShadowPrice</i>
Houston	320000	120.9384
Seattle	73000	0.0

Notice that **MPL** automatically uses the same name as the default for the index column *FactID*, as in the original tables the *factory* index was defined from. If an index column does have a different name than in the original table you can specify it following the table name by first entering the index name followed by an equal sign and the column name.

```
SUBJECT TO
  FactoryCapacity[factory]
  EXPORT ShadowPrice TO DATABASE("Factory","ShadowPrice"
                                factory="Factory") :
```

This means, if you are consistent in naming the columns in different tables, you do not have to specify them each time you refer to them in **MPL**.

MPL supports multiple databases for both importing and exporting data. The default database is specified in the *Database Options* dialog box in the *Options* menu. If you need to export a constraint vector to table in a database other than the default, you can do so by specifying the database name before the name of the table.

In the example below, **MPL** will export the shadow price of the constraint vector *FactoryCapacity* from **FoxPro** instead of the default database.

```
SUBJECT TO
  FactoryCapacity[factory]
  EXPORT ShadowPrice TO DATABASE(FoxPro,"Factory","ShadowPrice");
```

MPL also allows you to export constraint values other than the shadow price. You can export the slack values, the upper and lower ranges for the right-hand-side as well as the right-hand-side values. You change which values will be exported by entering one of the following keywords directly after the keyword *EXPORT*: *Activity*, *Slack*, *ShadowPrice*, *RhsValue*, *RhsLower*, *RhsUpper*. For example, if you want to export the slack for a constraint enter the following:

```
SUBJECT TO
  FactoryCapacity[factory]
  EXPORT Slack TO DATABASE("Factory","Slack");
```

If you need to export more than one value for a variable vector you can do so by entering multiple export statements after the variable definition.

MPL offers three different options on how the database table is updated by the *EXPORT* statement. The first one, which is the default, searches through the database and for each record locates the corresponding value in the solver solution and updates the database entry with it. This option minimizes the changes done to the database table since only the existing values are updated, but can sometimes be slow especially on SQL type databases.

The second option is to use the *REFILL* keyword right after the *EXPORT* keyword to specify that the whole database table should be emptied and then refilled with the entries from the solver solution. Since this takes out the necessity to search the table this can often lead to faster export times for larger tables.

The third option is to use the *CREATE* keyword right after the *EXPORT* keyword to specify that the database table should be created and then filled with the entries from the solver solution. This option is mainly useful when exporting to the database table for the first time.

Part III The MPL Modeling Language



PART IV

A MPL TUTORIAL

- Session 1: Running MPL on a Sample Model
- Session 2: Formulating a Simple Product-Mix Model
- Session 3: Introducing Vectors and Indexes
- Session 4: Planning Model with Multiple Time Periods
- Session 5: Planning Model with Multiple Plants
- Session 6: Allow Shipments Between Plants
- Session 7: Using Sparse Data in MPL Models

Part IV A MPL Tutorial



Tutorial Overview

In this chapter we will introduce you to the *MPL Modeling System* through a tutorial. This tutorial contains multiple sessions with a series of models, gradually increasing in difficulty; in order to explain how to formulate linear programming models. This tutorial is specifically designed for teaching optimization modeling the way it is being applied in the corporate world. By the end of this chapter, you should have a working knowledge of **MPL** and how to formulate models. The tutorial contains the following sessions:

- **Session 1: Running MPL on a Sample Model**

Session 1 introduces you to the *MPL Modeling System*, and how you can use the *Integrated Model Development Environment* to solve optimization problems. We show you how to start the **MPL** application and load a sample model, solve the model using one of the available optimizers, and then view the solution. Information on how to access the on-line help system in **MPL** will also be outlined. The purpose of this session is to give you an overview on how to solve models in **MPL** and to get you acquainted with the program. If you are already familiar with **MPL** and graphical user interfaces, such as Windows, you can go on to the next session, without losing continuity.

- **Session 2: Formulating a Simple Product-Mix Model in MPL**

In session 2, you will be introduced to the process of formulating linear programming models, by identifying the decision variables, the objective function and the constraints for the model. The session contains a description of a simple *product-mix model*, with two variables and three constraints. The purpose of this session is to have you use **MPL** through an example, by creating a simple model in order to understand the basic steps of formulating a model. Then you will solve the model and analyze the solution that is generated.

- **Session 3: Introducing Vectors and Indexes in MPL Models**

In session 3, you will learn the basics of how to use indexes and vectors to formulate models. You will see how indexes make it easy to quickly adjust the problem size. You will then find out how to use vectors to define model elements, such as data, variables, and constraints in a more efficient manner using the indexes. Finally, you will see how to use summations and macros on the vectors in your model formulation.

- **Session 4: A Production Planning Model with Multiple Time Periods**

In session 4, you will expand the model from the previous session, to include multiple time periods. A new index is introduced into the model to define time periods, and then you will update the various vectors in the model that are affected to account for the new index. You will become familiar with a new kind of constraint, called a *balance constraint*, that is used to connect together the production, sales and inventory variables for the model.

- **Session 5: Upgrade the Planning Model to Include Multiple Plants**

In session 5, you will encounter a model that has multiple plants available to produce the products. You will take the model from the previous session, and upgrade it to include another index *plant*, which will represent all of the plants. You will then go through the model, step by step, and update all the variables and constraints to account for the new index. Finally, you will learn how to use external data files to store data that becomes too large to be included in the actual model file.

- **Session 6: Upgrade the Model to Allow Shipments Between Plants**

In session 6, you will take the model from the previous session and upgrade it to allow shipments between the plants. This means that each plant can sell the products and maintain inventory independently, instead of doing it from a single source. To fulfill the demand in the most efficient manner it is necessary to be able to ship the products between the plants. Finally, you will learn how to use *where conditions* to remove the vector elements that are not valid, such as shipping back to the same location.

- **Session 7: Using Sparse Data in MPL Models**

In session 7, you will take the model from the previous session and add multiple machines for each plant. This will introduce a sparsity into the model, since not all machines are available in all the plants. You will use a new feature, a *sparse data vector*, to represent which machines are available in which plant. You will learn different ways on how to define sparse data vectors in **MPL**, including using the *IN* operator and sparse data files.

SESSION 1:

Running MPL on a Sample Model

In this session, you will go through a series of steps to learn how to run **MPL** on a sample model. By the end of the session, you will be able to start **MPL**, load and solve the model, and view the solution. At that point, you will learn how you can change option settings for **MPL** through dialog boxes. A brief outline of our **MPL** Help System is also covered at the end of this session. You will now go through each step in detail using the formulation of a sample model.

1.1 Your First MPL Session

There are four simple steps you need to be familiar with in order to solve models in **MPL**:

- Start **MPL**
- Load the model file
- Solve the model
- View the solution

Step 1: Start the MPL Application

Starting **MPL** is simple in Windows. When you installed **MPL**, the installation program created an entry in the *Start* menu under *Programs* for *MPL for Windows*. If **MPL** has not been installed, please refer to *Chapter 2* for information on how to install the software. To start **MPL**, click the *Start* button from the task bar that appears along the bottom of the screen and select *Programs / MPL for Windows*.

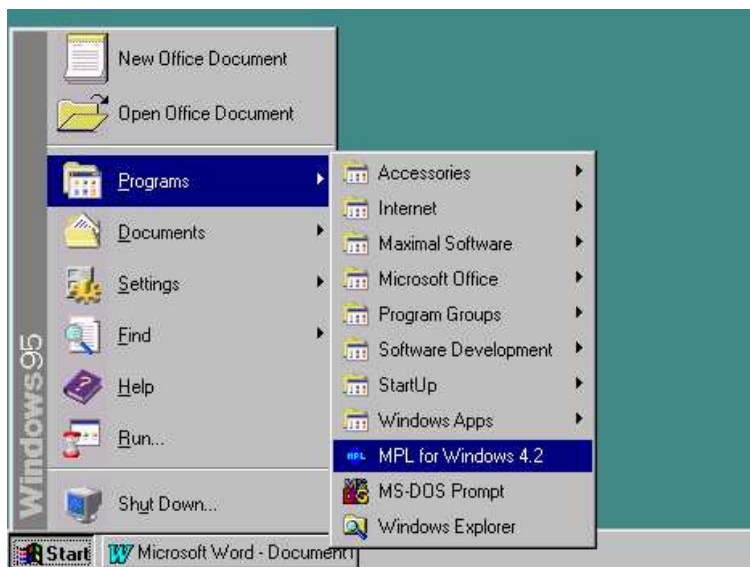


Figure T1.1: MPL for Windows in the Start Menu

Step 2: Load the Model File into MPL

After starting **MPL**, the next step is to load a model file in the model editor. The **MPL** application comes with several sample models that are installed in the *Mplwin4* directory. **MPL** models are stored as standard text files and typically have the file extension *.mpl*. The model we are using in this session is called *Model1.mpl* and is stored along with other models for this tutorial in a separate folder called *Tutorial*.

1. Choose *Open* from the *File* menu to display the *Open* dialog shown below.
2. Double click on the *Tutorial* folder name to go down to the folder where the model file *Model1.mpl* is stored.

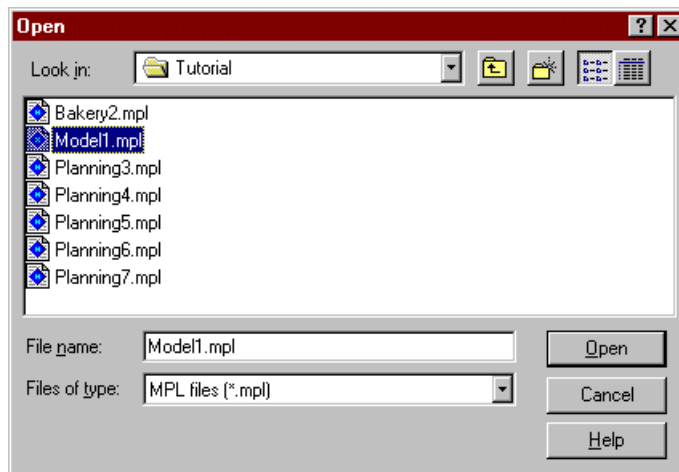
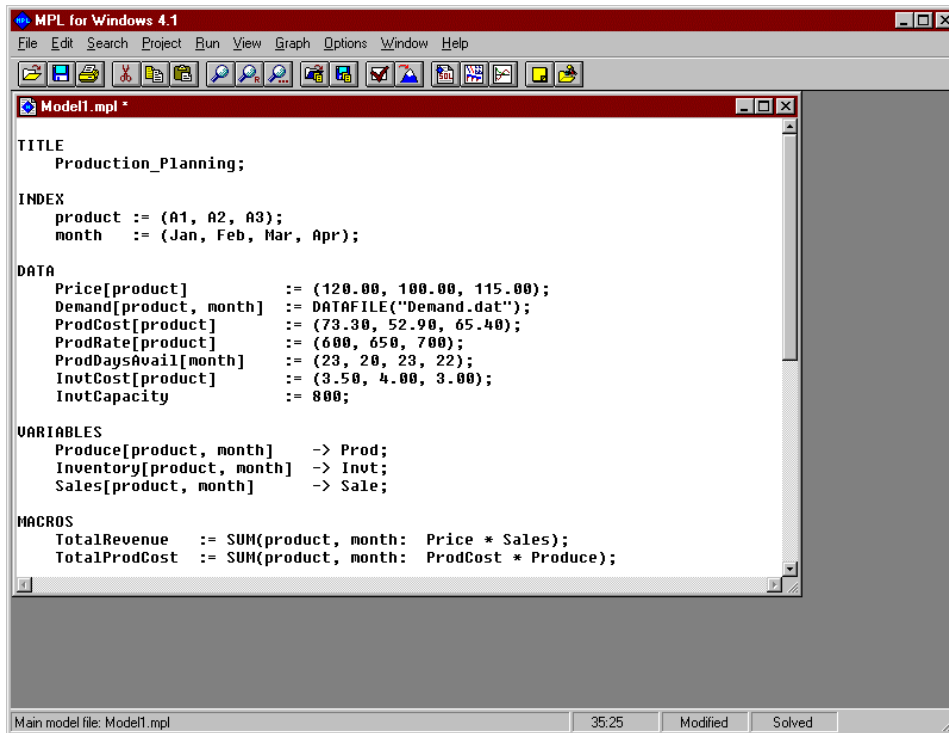


Figure T1.2: The Open File Dialog Box

3. The *Open* dialog now lists all the **MPL** model files in the *Tutorial* folder. Click on the filename *Model1.mpl* to select the model file and then press *Open* to open the file. Alternatively, you can open the file directly by double clicking on it in the list of files.

Part IV A MPL Tutorial

This will open a new model editor window containing the formulation for the model.



```

MPL for Windows 4.1
File Edit Search Project Run View Graph Options Window Help
Model1.mpl *
TITLE
  Production_Planning;
INDEX
  product := (A1, A2, A3);
  month   := (Jan, Feb, Mar, Apr);
DATA
  Price[product]           := (120.00, 100.00, 115.00);
  Demand[product, month] := DATAFILE("Demand.dat");
  ProdCost[product]        := (73.30, 52.90, 65.40);
  ProdRate[product]        := (600, 650, 700);
  ProdDaysAvail[month]     := (23, 20, 23, 22);
  InvtCost[product]        := (3.50, 4.00, 3.00);
  InvtCapacity             := 800;
VARIABLES
  Produce[product, month]  -> Prod;
  Inventory[product, month] -> Invt;
  Sales[product, month]    -> Sale;
MACROS
  TotalRevenue := SUM(product, month: Price * Sales);
  TotalProdCost := SUM(product, month: ProdCost * Produce);
Main model file: Model1.mpl      35:25      Modified      Solved

```

Figure T1.3: Model Editor Window with the *Model1.mpl* model file

Step 3: Solve the Model

In this tutorial you will be using the **CPLEX** solver, but if you have another solver installed that is supported by **MPL**, you can use that one instead.

When you run **MPL** for the first time after installing it, **MPL** will automatically try to locate any solvers that you have available. You can see which solvers have been set up for **MPL** by going to the *Run* menu. Each solver found will be listed in the menu with the word *Solve* in front of it.

If you do not have any solvers available you will see the item *No Solvers* in the *Run* menu. In this case, refer to *Chapter 2.3: Setting up Solvers for MPL* for information on how to add solvers to **MPL**. The rest of this tutorial will assume that you have successfully installed **CPLEX** or some other solver and it has been set up correctly.

The next step is to solve the model that you have loaded into the model editor. To solve the model follow these steps:

1. Choose *Solve CPLEX* from the *Run* menu to solve the *Model1* model.
2. While solving the model, the *Status Window* is displayed to provide you with information about the solution progress.

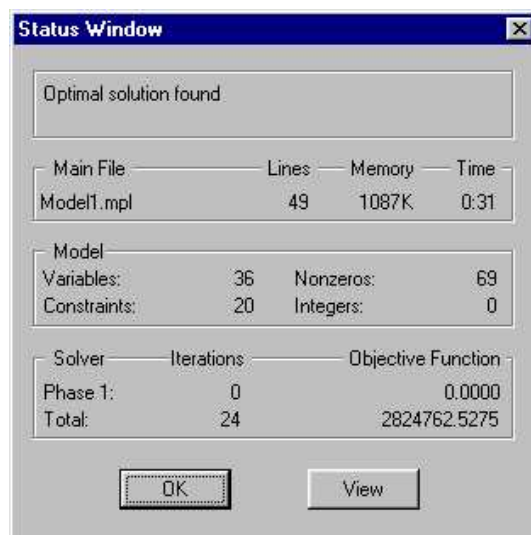


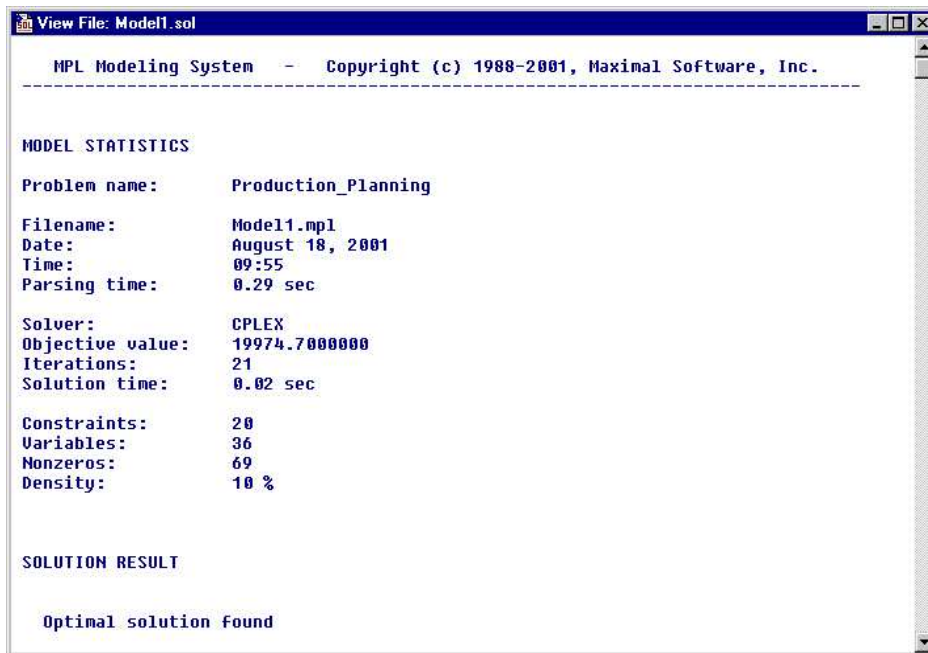
Figure T1.4: The Status Window

The *Status Window* provides you with information such as the number of lines read, the number of variables and constraints encountered, and how much memory has been used. While the optimizer is solving the model, the number of iterations and the current value of the objective function are also shown. The figure above shows the *Status Window* after the model has been solved.

Step 4: View the Solution

MPL automatically sends the solution to a file with same name as the model file, but with the file extension `.sol`. Use the following steps to view the solution file `Model1.sol` that was generated for the model you just solved:

1. Press the *View* button at the bottom of the *Status Window*, which appeared on the screen during the solving process. This will open a *View Window* containing the solution file as shown below:



```
View File: Model1.sol
MPL Modeling System - Copyright (c) 1988-2001, Maximal Software, Inc.
-----
MODEL STATISTICS
Problem name:      Production_Planning
Filename:          Model1.mpl
Date:              August 18, 2001
Time:              09:55
Parsing time:      0.29 sec

Solver:            CPLEX
Objective value:   19974.7000000
Iterations:        21
Solution time:     0.02 sec

Constraints:       20
Variables:         36
Nonzeros:          69
Density:           10 %

SOLUTION RESULT

Optimal solution found
```

Figure T1.5: View Window with the Solution File `Model1.sol`

2. You can quickly browse through the solution using the scroll bars on the right. Note that the details of the solution are provided including the optimal solution value for the objective function, values for the decision variables and for the constraints.
3. When you are finished browsing through the solution you can press the (X) button in the upper right hand corner to close the *View Window*.

Step 5: Using the Model Definitions Tree Window

MPL also allows you to view all of the defined items from the model formulation in a hierarchical tree window called the *Model Definitions Window*. Each branch corresponds to a section in the model.

If the tree window is not open, you can open it by choosing *Model Definitions* from the *View* menu. While working in **MPL** it is normally a good idea to leave the tree window open at all times. **MPL** will automatically update its contents every time you solve your model.

The tree window provides easy access to the different areas of the model and allows you to quickly view selected parts of the solution for the model. To use the tree window do the following:

1. Make sure the tree window is open by choosing *Model Definitions* from the *View* menu.

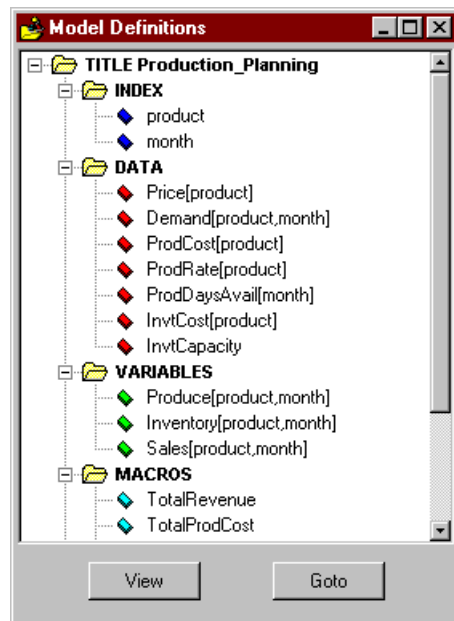
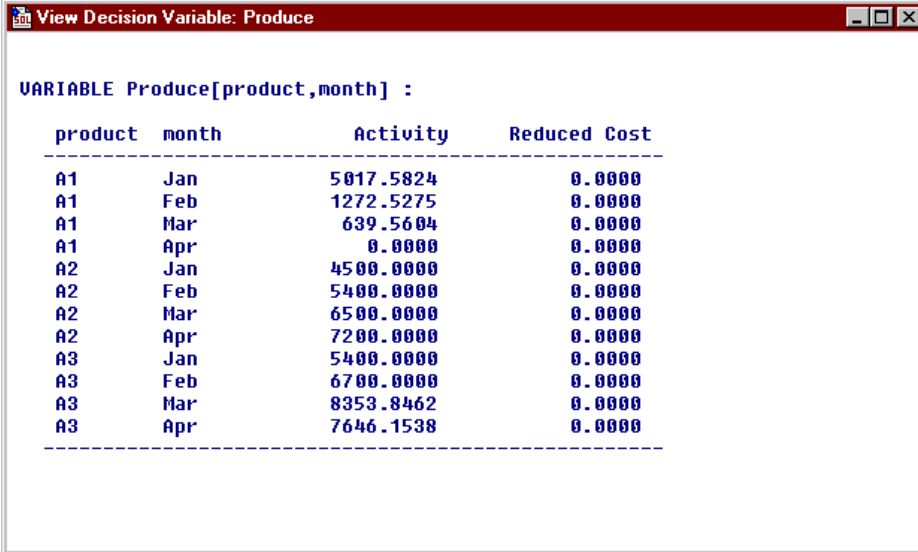


Figure T1.6: The Model Definitions Tree Window

2. Under the header *VARIABLES* in the tree window, you will see three variable names listed, *Produce*, *Inventory*, and *Sales*; which are the variables for the model. In front of each section header, there is a small box containing either a plus or a minus sign. This box allows you to quickly expand and collapse each branch of the tree.

3. Now select the variable *Produce* and press the *View* button at the bottom of the window. This will open a new *View Window* with the solution values for the *Produce* variable only.



View Decision Variable: Produce

VARIABLE Produce[product,month] :

product	month	Activity	Reduced Cost
A1	Jan	5017.5824	0.0000
A1	Feb	1272.5275	0.0000
A1	Mar	639.5604	0.0000
A1	Apr	0.0000	0.0000
A2	Jan	4500.0000	0.0000
A2	Feb	5400.0000	0.0000
A2	Mar	6500.0000	0.0000
A2	Apr	7200.0000	0.0000
A3	Jan	5400.0000	0.0000
A3	Feb	6700.0000	0.0000
A3	Mar	8353.8462	0.0000
A3	Apr	7646.1538	0.0000

Figure T1.7: View Window with the Solution Values for the *Produce* Variable

1.2 Using the Help System in MPL

MPL offers the user an extensive help system containing useful information on how to use the software.

Accessing the Available Help Topics for MPL

To open the main help system window for **MPL**, go to the *Help* menu and choose *Topics*. This will display the help window, shown below, where you can select the help topic you wish to read.

The *MPL Help Topics dialog box* contains three *tabs*; the *Contents* tab, the *Index* tab, and the

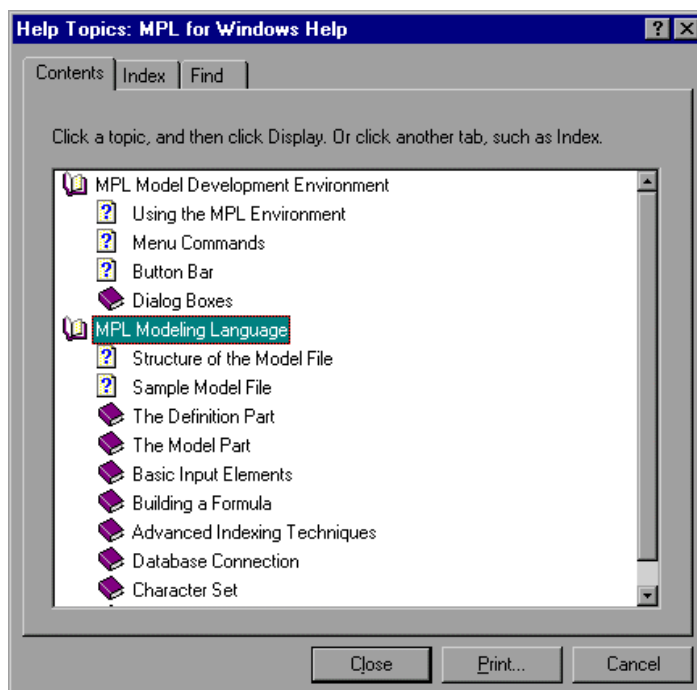



Figure T1.8: The Help Contents Window for MPL

Find tab, giving you different ways of accessing help.

The *Contents* tab, shows all the available topics in the help system in a hierarchical tree structure. The *Index* tab, enables the user to access a list of all the keywords in the help file or to perform a search for a specific keyword. The *Find* tab, enables the user to find a help topic by searching for specific words or phrases in the text. The Find database is built the first time you select the *Find* tab through the *Find Setup Wizard*.

Looking Up Context Sensitive Help for Dialog Boxes

One of the most beneficial features in Windows is the context sensitive help for dialog boxes. This feature enables you to quickly get specific help information about the dialog box item you are interested in. To learn how to use the context sensitive help in **MPL** follow these steps:

1. To open a dialog box choose *MPL Language* from the *Options* menu. This will display the *MPL Language Options* dialog box.
2. Click on the question mark button  in the upper right corner of the dialog and then release the mouse button. The cursor will change to a (?) which you can use to point to an item you are interested in..
3. Move the mouse across the dialog and click on the *Max subscript length* item. A small window will pop up with a short explanation of the item you selected as shown here below.

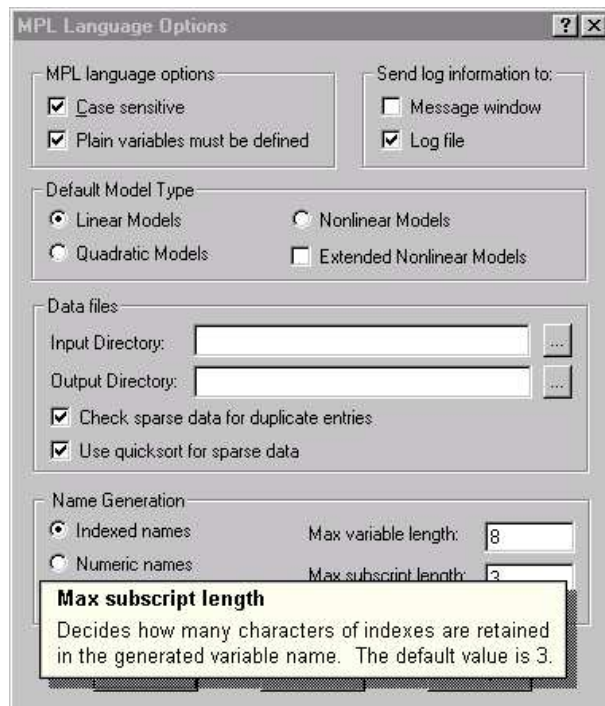


Figure T1.9: Context Sensitive Help for Dialog Box

You can also display the context sensitive help by pressing the right mouse button on the item in the dialog box and then select *What's This?* from the pop up menu.

SESSION 2:

Formulating a Simple

Product-Mix Model

In this session you will be introduced to the formulation of linear programming models through a simple product-mix problem called **Better Bread Bakery**.

The purpose of this session is to introduce to you the basic concepts of:

- Decision variables
- Objective function
- Constraints

You need to identify these concepts when formulating linear programming models.

Part IV A MPL Tutorial

The first step when formulating a model is to identify and give names to the *decision variables*. Decision variables are the elements of the model that the decision maker controls and those values determine the solution of the model.

The next step is to determine the *objective function* in terms of the decision variables. The *objective function* is where you specify the goal you are trying to achieve. The goal can either be to maximize or to minimize the value of the objective function. We sometimes use the phrase that we want to *optimize* the model. This means we want to find the values for the decision variables which gives either the maximum or the minimum value of the objective function. In many cases, the objective function has a monetary value, for example to maximize profit or minimize cost, although this is not always the case.

Constraints are the real world limitations on the decision variables. A constraint restricts or constrains the possible values which the variable can take. An example of a constraint could be, for example, that certain resources, such as machine capacity or manpower, are limited.

2.1 Problem Description: A Simple Product-Mix Model

The **Better Bread Bakery** is famous for its breads. They make two kinds: “*Sunshine*”, a white bread and “*Moonlight*”, a large dark bread.

The market for the famous breads is endless. Every *Sunshine* loaf sold brings a profit of \$0.05 and each loaf of *Moonlight* breads brings a profit of \$0.08. There is a fixed cost of running the bakery of \$4000 per month, regardless of the amount of bread baked.

The bakery is divided into two departments: baking and mixing, with limited capacity in both departments.

In the baking department there are ten big ovens, each with a capacity of 140 baking sheets per day. It is possible to put ten loaves of *Sunshine* on each of these baking sheets, or 5 of the larger *Moonlight* breads. You can make any combination of the two breads on the baking sheets. Just keep in mind that each *Moonlight* loaf takes twice the space of a *Sunshine* loaf.

The mixing department can mix up to 8000 loaves of *Sunshine* per day and 5000 loaves of *Moonlight* bread. There are two separate automatic mixers so there is no conflict between making the two kinds of dough.

Since the market for both types of breads is unlimited, the management of **BBB** has decided to find the best product mix. The question is how many loaves of each type of bread should be baked each day to produce the highest profit, given the physical limitations of the bakery.

We will now show you how to identify the decision variables, the objective function and the constraints for this model and then enter the formulation in **MPL**.

2.2 Formulating the Model

Identify the Decision Variables

For our bakery, the decision variables correspond to the number of loaves of each type made daily. To make the formulation easier to read, it is a good idea to give the decision variables names, which allow you to identify what they represent in the real world. Use two decision variables, named *Sun* and *Moon*, and agree that they have the following meanings:

Sun = The number of loaves of Sunshine bread produced per day

Moon = Number of loaves of Moonlight bread produced per day

Now you want to determine the values for these two decision variables in order to maximize the bakery's profit.

Identify the Objective Function

In our example the goal is to maximize the daily profit. We make a profit of \$0.05 on each *Sunshine* loaf, so the total daily production of *Sunshine* bread yields \$0.05 multiplied by the value of the *Sun* variable for profit.

For the *Moonlight* production, the corresponding yield is \$0.08 multiplied by the value of the *Moon* variable. We call the values \$0.05 and \$0.08 the *coefficients* for the corresponding decision variables in the objective function. To get the total contribution towards profit in a day, we add the contributions from the two bread types. From that, we subtract the fixed cost of \$4000 divided by 30 days in a month, to obtain the net daily profit. This leads to the following quantity we want to maximize:

$$\text{Profit} = 0.05 \text{ Sun} + 0.08 \text{ Moon} - 4000/30$$

We have now defined the objective function for this particular problem. The solver uses the objective function as the criteria to determine which solution is optimal.

Identify the Constraints

The first constraint in the baking department is somewhat complicated since there is an interaction between the bread types. It is possible to put either ten *Sunshine* breads or five *Moonlight* breads on each baking sheet. It is also possible to use any combination of the two. The expression $1/10 \text{ Sun} + 1/5 \text{ Moon}$ gives us the total usage of baking sheets. If you measure the capacity of each oven as the number of baking sheets which it can handle per day (10×140), you can express the constraint as:

$$1/10 \text{ Sun} + 1/5 \text{ Moon} \leq 10 \times 140$$

We express the constraints which were given by the mixing department like this:

$$\begin{aligned} \text{Sun} &\leq 8000 \\ \text{Moon} &\leq 5000 \end{aligned}$$

Summing up the Formulation

We have now defined the variables the objective function, and all of the constraints. This is the *formulation* of the linear programming problem as shown below:

Max

$$\text{Profit} = 0.05 \text{ Sun} + 0.08 \text{ Moon} - 4000/30$$

Subject to

$$\begin{aligned} 1/10 \text{ Sun} + 1/5 \text{ Moon} &\leq 10 \times 140 \\ \text{Sun} &\leq 8000 \\ \text{Moon} &\leq 5000 \end{aligned}$$

Once you have your formulation, most of the work is done. As you are about to see, **MPL** accepts its input in a form very similar to what you have just written down.

2.3 Solving the Model in MPL

Step 1: Start MPL and Create a New Model File

1. Start the **MPL** application.
2. Choose *New* from the *File* menu to create a new empty model file.
3. Choose *Save As* from the *File* menu and save the file as *Bakery2.mpl*.

Step 2: Enter the Model Formulation for the Bakery Model

You are now ready to enter the model into the **MPL**. The model editor in **MPL** is a standard text editor which allows you to enter the model and perform various editing operations on the model text. In the model editor, enter the following model formulation:

```
TITLE BetterBreadBakery;
MAX
    Profit = 0.05 Sun + 0.08 Moon - 4000/30 ;
SUBJECT TO
    1/10 Sun + 1/5 Moon <= 10 * 140 ;
    Sun <= 8000 ;
    Moon <= 5000 ;
END
```

Notice that there is one small difference between the formulation in the previous step and the file shown here. There is a semicolon ‘;’ after the objective function and after each constraint. This allows **MPL** to separate the constraints.

The spacing used between entries and lines in **MPL** is not rigid. It is recommended, when entering the model, to use spaces and extra lines to make the model formulation easier to read and understand. **MPL** is only concerned with the actual text in the model file.

When you have finished entering the model choose *Save* from the *File* menu to save the model.

Step 3: Check the Syntax of the Model

After you have entered the formulation in the model editor, you can check the model for syntax errors. If **MPL** finds a mistake in the formulation it will report it in the *Error Message* window showing the erroneous line in the model, along with a short explanation of the problem. The cursor is automatically positioned at the mistake in the model file, with the offending word highlighted.

To check the syntax at the model choose *Check Syntax* from the *Run* menu. If there are no errors found **MPL** will respond with a message stating that the syntax of the model is correct. If there is an error in the model **MPL** will display the *Error Message* window.

If you did not have any errors in your formulation (*congratulations!*) you may still want to see how the error messages work. We are going to introduce an error in the model and see how the error messages in **MPL** can help you correct it.

1. In the model editor remove the semicolon at the end of the first constraint as follows:

```
SUBJECT TO
    1/10 Sun + 1/5 Moon <= 10 * 140    ! note the missing semicolon
    Sun <= 8000 ;
    Moon <= 5000 ;
```

2. Choose *Check Syntax* from the *Run* menu. **MPL** will go through the model and find the missing semicolon when it is parsing the second constraint displaying the following error message:



Figure T2.1: The Error Message Window


Part IV A MPL Tutorial

The reason **MPL** did not notice the missing semicolon until it reached the second constraint is because it thinks $10*140$ is a coefficient for the *Sun* variable in the line below.

3. When you press the *OK* button you are returned to the model editor. The cursor will automatically be positioned at the location where **MPL** found the error which in our case is at the ' \leq ' in the second constraint.
4. Now you can reenter the semicolon for the first constraint and if you check the syntax again, **MPL** will report back with message that the syntax is correct.

Step 4: Solve the Model

The next step is to solve the *Bakery2* model. Solving the model involves several tasks for **MPL**, including checking the syntax, parsing the model into memory, transferring the model to the solver, solving the model and then retrieving the solution from the solver and creating the solution file. All these tasks are done transparently to the user when he chooses the solve command from the menus. To solve the model follow these steps:

1. Choose *Solve CPLEX* from the *Run* menu or press the *Run Solve*  button in the *Toolbar*.
1. While solving the model the *Status Window*; appears providing you with information about the solution progress.

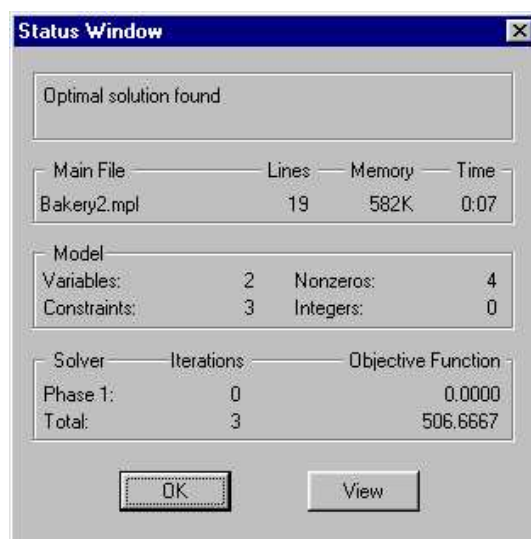


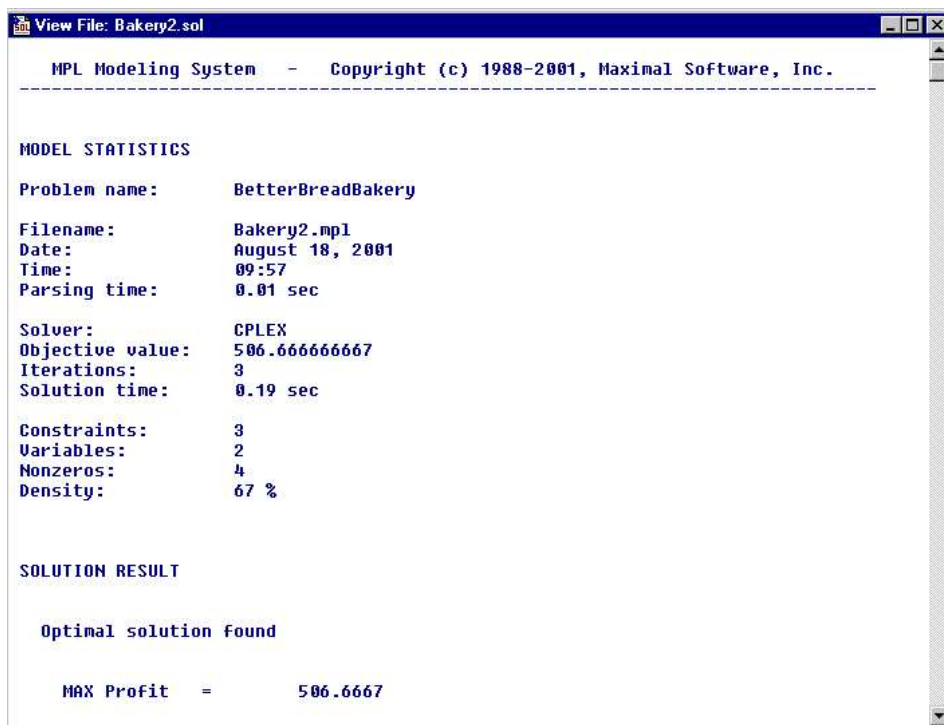
Figure T2.2: The Status Window for the *Bakery2* model

If everything goes well **MPL** will display the message “*Optimal Solution Found*”. If there is an error message window with a syntax error please check the formulation you entered with the model detailed earlier in this session.

Step 5: View and Analyze the Solution

After solving the model **MPL** automatically creates a standard solution file containing various elements of the solution to the model. This includes among other things the optimal value of the objective function, the activity and reduced costs for the variables, and slack and shadow prices for the constraints. This solution file is created with the same name as the model file but with the extension *.sol*. In our case the solution file will be named *Bakery2.sol*.

After you have solved the model you can display the solution file in a view window by pressing the *View* button at the bottom of the *Status Window*. This will display the view window shown below.



```
View File: Bakery2.sol
-----
MPL Modeling System - Copyright (c) 1988-2001, Maximal Software, Inc.
-----
MODEL STATISTICS

Problem name:      BetterBreadBakery
Filename:         Bakery2.mpl
Date:            August 18, 2001
Time:           09:57
Parsing time:    0.01 sec

Solver:          CPLEX
Objective value: 506.66666667
Iterations:     3
Solution time:  0.19 sec

Constraints:     3
Variables:      2
Nonzeros:       4
Density:        67 %

SOLUTION RESULT

Optimal solution found

MAX Profit = 506.6667
```

Figure T2.3: View Window with the *Bakery2.sol* Solution File

The *View Window* stores the solution file in memory, allowing you to quickly browse through the solution using the scroll bars. A full listing of the solution file is shown on the next page.

Session 2 Formulating a Simple Model

MPL Modeling System - Copyright (c) 1988-2001, Maximal Software, Inc.

MODEL STATISTICS

Problem name: BetterBreadBakery
Filename: Bakery2.mpl
Date: April 17, 1998
Time: 17:29
Parsing time: 0.04 sec
Solver: CPLEX
Objective value: 506.66666667
Iterations: 3
Solution time: 0.14 sec
Constraints: 3
Variables: 2
Nonzeros: 4
Density: 67 %

SOLUTION RESULT

Optimal solution found

MAX Profit = 506.6667

DECISION VARIABLES

PLAIN VARIABLES

Variable Name	Activity	Reduced Cost
Sun	8000.0000	0.0000
Moon	3000.0000	0.0000

CONSTRAINTS

PLAIN CONSTRAINTS

Constraint Name	Slack	Shadow Price
c1	0.0000	-0.4000
c2	0.0000	-0.0100
c3	2000.0000	0.0000

END

Part IV A MPL Tutorial

The first part of the solution file contains various statistics for the model, such as the filename, date and time the model was solved, which solver was used, the value of the objective function and the size of the model.

The next part of the solution file contains the solution results. Here you can see if the solution that was found was optimal or if it was unbounded or infeasible. It also shows you the name and optimal value of the objective function. In our case the profit for the bakery is equal to \$506 per day.

In the *DECISION VARIABLES* section you get a list of the variables in the model, *Sun* and *Moon*. You will see that for *Sun* bread the solution suggests that you produce 8000 loaves per day, which is the same amount as the capacity of the mixing department for the *Sun* bread. For the *Moon* bread the solution suggests that we produce 3000 loaves per day, which is less than the maximum capacity of 5000 *Moon* loaves per day in the mixing department.

In the *CONSTRAINTS* section the solution file lists all of the constraints for the model. In our model we had three constraints, one for the ovens in the baking department, and two constraints in the mixing department for each type of bread. Since the slack for the first constraint is zero this means that the ovens in the baking department are running at full capacity. In a similar way, the mixers for the *Sun* bread are running at full capacity, but the mixer for the *Moon* bread has a slack of 2000.

Therefore, since the ovens in the bakery department are shared between the two types of breads, the solver has chosen to produce as much of the *Sun* bread as possible, then use the rest of the capacity to produce the *Moon* bread.

SESSION 3:

Introducing Vectors and Indexes

in MPL Models

MPL allows you to concisely express multitudes of vectors, data, and constraints, with intuitive and flexible expression formats. In Session 2, you were introduced to plain variables, the objective function and plain constraints. In this Session we introduce several new model elements that are necessary for constructing models for larger problems.

The model you were using in Session 2 was small, involving only two variables. Real world models have hundreds or thousands of variables and constraints, and sometimes extend to millions of variables. **MPL** allows the user to set up a model using *Indexes*, *Data Vectors*, *Vector Variables*, and *Vector Constraints* to define the problem in a concise, easy-to-read way.

3.1 New Concepts in this Session

Indexes as the Domains of the Model

Indexes define the domains of the model, encapsulate the problem dimensions, and make it easy to quickly adjust the problem size. Once you have defined the indexes for a model, use them to define data, variable, and constraint vectors.

The realm of subscripted variables and constraints is where a modeling language, such as **MPL**, can allow dramatic productivity, since it allows the model developer to formulate the model in a concise, easy to read manner, using indexes and vectors. Examples of indexes include:

- products
- months
- plants

Data, Variable, and Constraint Vectors

Vectors are basically aggregations of elements in the model that share common characteristics and purpose. Once you have defined the indexes in a model, you can use them to define vectors that contain the data, variables and constraints for the model. This allows you to work in a more condensed way, as you don't have to type every element each time you need it.

Data Vectors are used when the coefficients for the problem come in lists or tables of numerical data. When an index is defined there is one value for each element of the index and the data vectors allow you to group collections of data together in the model. This data can either be specified as lists of numbers in the model file, or retrieved from an external file, which will be covered in the next session. Examples of data vectors, in a production model with a 'product' index, include:

- Price for each product
- Production cost for each product
- Demand for each product

Variable Vectors can be defined in a similar way as data vectors, to form a collection of variables defined over a certain index. Examples of variable vectors include:

- How much to produce of each product
- Inventory level of the product in each month
- How much to ship between locations or plants

Session 3 Introducing Vectors and Indexes

Constraint Vectors are defined over indexes, which **MPL** expands to a collection of simple constraints when generating the model. A vector constraint can be defined in this way, over a number of indexes such as periods and products. Examples of constraint vectors include:

- Limit of production to capacity
- Limit how much you sell to what the demand is.

Data Constants

Data Constants are used in the model to aid readability, and make the model easier to maintain. They are assigned a specific value, but not defined over a specific index.

Using Summations over Vectors

One of the operations usually done on vectors is to sum or add all the values for each element of the vector. This is done in **MPL** by using the keyword **SUM** surrounding the vector expression to be added together. The expression is prefixed by a list of indexes, over which the sum hinges. The sum expression normally contains a single variable vector per term, possibly multiplied by one or more data vectors.

```
SUM(product: Price * Sales);  
SUM(product, month: ProdCost * Produce);
```

3.2 Problem Description: A Product-Mix Model with Three Variables

You are now going to create a model formulation to see how indexes and vectors are used in **MPL**. We formulated a small *product-mix* model for a bakery that contained two products and three constraints. In this example, you are going to do a similar model, but this one will include three products, which will be called *A1*, *A2* and *A3*. For these three products you are going to create an index, and then create a variable vector that represents how much of each of these products need to be produced.

Given the definitions of these new terms, indexes and vectors, you are going to apply them to a sample model. The *product-mix* model is a question of how to distribute production capacity between products, and to determine the production level, given the demand.

The *selling price* for each product is fixed. \$120.00 for *A1*, \$100.00 for *A2*, and \$115.00 for *A3*. There is also a limit on the *maximum demand* for each product 4300 for *A1*, 4500 for *A2*, and 5400 for *A3*.

The *production rate* is measured by how many items of each product are produced each day. In this problem, you have a total of 22 production days available. The *production cost* is different for each product. The production rate and production cost for each product is given in the table below:

<i>Production</i>	<i>A1</i>	<i>A2</i>	<i>A3</i>
Production Cost	\$73.30	\$52.90	\$65.40
Production Rate	500	450	550

3.3 Formulation of the Model in MPL

The next step is to take the problem described in the previous session, and formulate it into the **MPL** model. To see an overview of the *Planning3* model to be created, a full listing of the model formulation, is shown below.

```
TITLE
  Production_Planning3;
INDEX
  product := (A1, A2, A3);

DATA
  Price[product]      := (120.00, 100.00, 115.00);
  Demand[product]     := (4300, 4500, 5400);
  ProdCost[product]   := (73.30, 52.90, 65.40);
  ProdRate[product]   := (500, 450, 550);
  ProdDaysAvail       := 22;

VARIABLES
  Produce[product] -> Prod;

MACROS
  TotalRevenue := SUM(product: Price * Produce);
  TotalCost    := SUM(product: ProdCost * Produce);

MODEL

  MAX Profit = TotalRevenue - TotalCost;

SUBJECT TO
  ProdCapacity -> PCap:
    SUM(product: Produce / ProdRate) <= ProdDaysAvail;

BOUNDS
  Produce <= Demand;

END
```

3.4 Enter the Model in MPL Step-by-Step

Step 1: Start MPL and Create a New Model File

You will now go through entering a simple model, step by step, to allow you to understand model formulation as you set it up.

1. Start the **MPL** application.
2. Choose *New* from the *File* menu to create a new empty model file.
3. Choose *Save As* from the *File* menu and save the file as *Planning3.mpl*.

Step 2: Specify a Title for the Model

The title is optional, but a convenient place to name the model. The title will be used in the solution file to identify the model. You should now have an empty editor window where you can enter the **MPL** formulation. To enter the title for the *Planning3* model, type in the following text in the model editor:

```
TITLE
  Production_Planning3;
```

Step 3: Define an Index for all the Products in the Model

The first section in an **MPL** model is usually the *INDEX* section where you define the indexes for the model. In this example, you have three products, *A1*, *A2*, and *A3* for which you are creating an index called *product*. In the model editor, directly below the title, add an *INDEX* section with a definition for the *product* index as follows:

```
INDEX
  product := (A1, A2, A3);
```

Step 4: Define the Data for the Model

The next section in **MPL** is usually the *DATA* section where you define the *Data Vectors* and *Data Constants* for the model. The first data vector you will enter, contains the prices for each product, which were given in the problem description.

In the model editor, directly below the index definition, add a *DATA* section with a definition for the data vector *Price* followed by the index *[product]* inside brackets:

```
DATA
  Price[product] := (120.00, 100.00, 115.00);
```

Following the declaration, you enter an assignment symbol ‘:=’ and then a list of numbers containing the prices for each product. Surround the list with parentheses and separate each number by either a space, comma or both. There should be a semicolon after each data vector definition to separate it from the other definitions in the model.

The problem description also listed data for the demand, production cost and production rate. There is a certain demand for each product, an amount it costs to produce, and an upper limit on how many of each product you can produce per day. To enter this data into **MPL** add the following definitions to the *DATA* section directly below the *Price* data vector:

```
Demand[product]   := (4300, 4500, 5400);
ProdCost[product] := (73.30, 52.90, 65.40);
ProdRate[product] := (500, 450, 550);
```

The problem description also listed how many production days there are available. Add the following data constant definition for the production days available directly below the *ProdRate* data vector:

```
ProdDaysAvail    := 22;
```

Step 5: Define a Variable Vector for How Much to Produce of each Product

Usually, the next section will be the *VARIABLES* section where you define the variables for the model. In the problem description, you were asked to determine how much to produce of each product. To do this you need to define a vector variable named *Produce* over the index *product*. In the model editor, directly below the data definitions, add the *VARIABLES* section with a definition for the *Produce* vector variable as follows:

```
VARIABLES
  Produce[product] -> Prod;
```

The name that appears after the ‘->’ (read becomes) sign is an optional abbreviation of the vector name used to offset the name size limitations of most LP solvers. This allows you to use long and descriptive names for the variables in your model.

Step 6: Define the Objective Function as Total Revenue Subtracted by Total Production Cost

In the problem description, you were asked to maximize the profit for the company, which is represented as *Total revenue - Total cost*. The total revenue is calculated by multiplying the price for each product by how much of that product is produced. In the same fashion, the total cost is calculated by multiplying the production cost for each product by the amount produced. These summations will be used to define the *objective function* for the model.

When entering *summations* for the objective function it is useful to define them separately as *macros*, to make the model easier to read. You can then, in the model, refer to these summations using the macro names. In the model editor, directly below the variable definition, enter the following macro definitions in the *MACROS* section.

```
MACROS
  TotalRevenue := SUM(product: Price * Produce);
  TotalCost    := SUM(product: ProdCost * Produce);
```

The model part in **MPL** is where you define the actual *objective function* and the *constraints* for the model. You will be using the macros, defined above, to create the objective function by referring to the macro names where you need to use the summations. Since you are maximizing the profit in this model, the name of this objective function will be *Profit*. In the model editor, enter the keyword *MODEL* to note the start of the model part, followed by the definition for the objective function:

```
MODEL
  MAX Profit = TotalRevenue - TotalCost;
```

The formula for the objective function is quite simple, as we are using macros to contain to actual summations. This results in the objective function determining how to maximize profit by computing the difference between the revenues and total cost.

Step 7: Enter a Constraint for the Production Capacity

Following the objective function you will define the *constraints* for the model in the *SUBJECT TO* section. In the problem description, you were given the *production rate* defined as how many items of each product you can produce each day. As well as how many production days are available. Since this limits how many items you can produce you will need to create a production capacity constraint called *ProdCapacity*, for each product.

In the model editor, add the *SUBJECT TO* heading, followed by the constraint definition:

```
SUBJECT TO
  ProdCapacity -> PCap:
    SUM(product: Produce / ProdRate) <= ProdDaysAvail;
```

In the summation, you divide the number of items produced by the production rate to receive the total number of days used to produce each product. The total production days must be less than the days available for production.

Step 8: Enter an Upper Bound for the 'Produce' Variable

The *BOUNDS* section is used to define the upper and lower bounds on the variables in the model. Bounds are similar to constraints but are limited to only one variable. In the problem description, we specified that the total number of items produced must be less than the demand. Therefore, enter the following upper bound on the *Produce* variable in the *BOUNDS* section.

```
BOUNDS
  Produce <= Demand;

END
```


Please note that in most linear programming models all variables have an implied lower bound of zero. These lower bounds are handled automatically by MPL and do not have to be specified unless they are nonzero.

At the end of the model enter the keyword *END* to note the end of the model. After you have finished entering the model, you should save it by choosing *Save* from the *File* menu.

3.5 Solve the Model and Analyze the Solution

Solve the Model

The next step is to solve the *Planning3* model. Solving the model involves several tasks done automatically by **MPL**, including checking the syntax, parsing the model into memory, transferring the model to the solver, solving the model and then retrieving the solution from the solver and creating the solution file. All these tasks are done transparently to the user when he chooses the solve command from the menus. To solve the model do the following steps:

1. Choose *Solve CPLEX* from the *Run* menu or press the *Run Solve*  button in the *Toolbar*.
1. While solving the model the *Status Window* appears providing you with information about the solution progress.

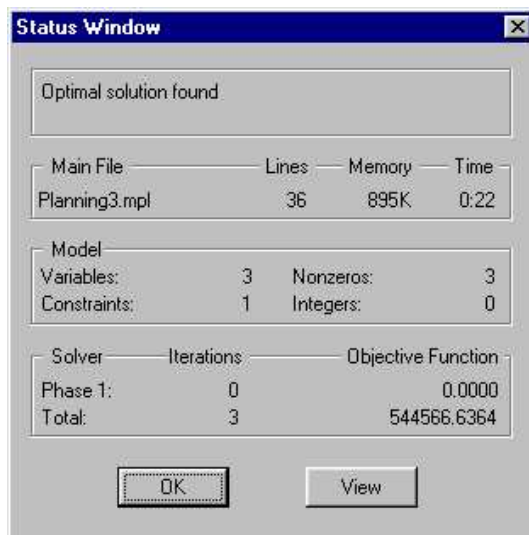


Figure T3.1: The Status Window for the *Planning3* model

If everything goes well **MPL** will display the message “*Optimal Solution Found*”. If there is an error message window with a syntax error please check the formulation you entered with the model detailed earlier in this session.

View and Analyze the Solution

After solving the model **MPL** automatically creates a standard solution file containing various elements of the model solution. This includes, among other things, the optimal value of the objective function, the activity and reduced costs for the variables, and slack and shadow prices for the constraints. This solution file is created with the same name as the model file, but with the extension *.sol* instead. In our case the solution file is named *Planning3.sol*.

After you have solved the model you can display the solution file in a view window by pressing the *View* button at the bottom of the *Status Window*. This will display the view window shown below.

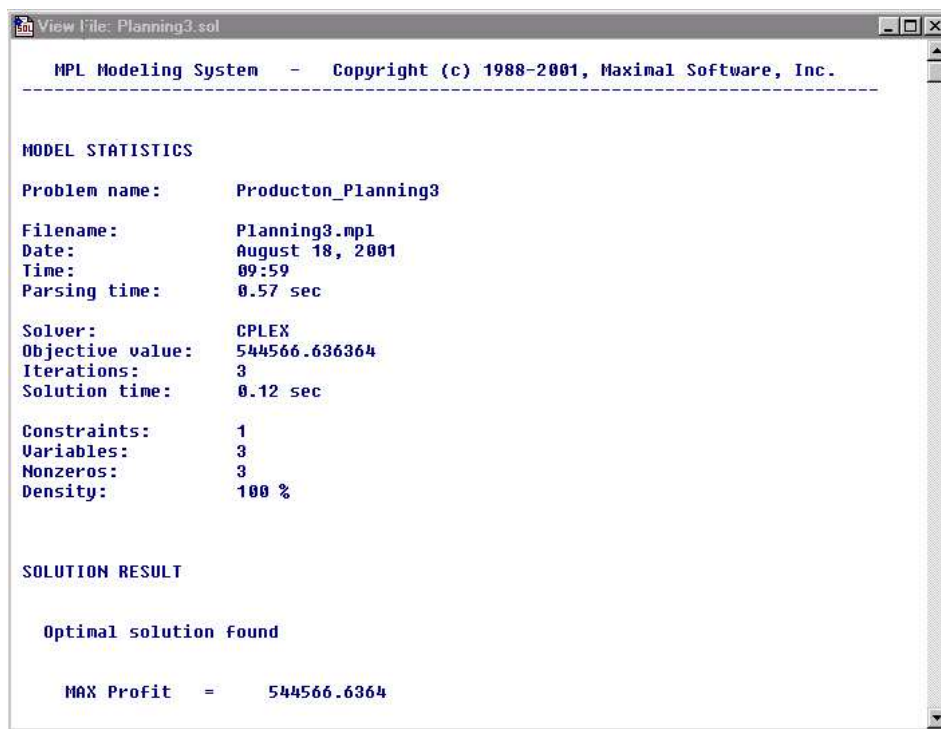


Figure T3.2: View Window with the *Planning3.sol* Solution File

The *View Window* stores the solution file in memory, allowing you to quickly browse through the solution using the scroll bars. A full listing of the solution file for the *Planning3* model is shown here on the next page.

Part IV A MPL Tutorial

MPL Modeling System - Copyright (c) 1988-2001, Maximal Software, Inc.

MODEL STATISTICS

Problem name: Producton_Planning3
Filename: Planning3.mpl1
Date: April 18, 1998
Time: 09:59
Parsing time: 0.57 sec

Solver: CPLEX
Objective value: 544566.636364
Iterations: 3
Solution time: 0.12 sec

Constraints: 1
Variables: 3
Nonzeros: 3
Density: 100 %

SOLUTION RESULT

Optimal solution found

MAX Profit = 544566.6364

MACROS

Macro Name	Values
TotalRevenue	1298181.8182
TotalCost	753615.1818

DECISION VARAIBLES

VECTOR Produce[product] :

product	Activity	Reduced Cost
A1	4300.0000	4.3100
A2	1611.8182	0.0000
A3	5400.0000	11.0636

CONSTRAINTS

PLAIN CONSTRAINTS

Constraint Name	Slack	Shadow Price
ProdCapacity	0.0000	-21195.0000

END

Session 3 Introducing Vectors and Indexes

The first part of the solution file contains various statistics for the model such as the filename, date and time the model was solved, which solver was used, the value of the objective function and the size of the model.

The next part of the solution file contains the solution results. Here you can see if the solution that was found was optimal, or if it was unbounded or infeasible. It also shows you the name and optimal value of the objective function.

In the *MACROS* section of the solution file you get a list of all of the macros defined in the model along with the solution values for them. For example, in our *Planning3* model, the total revenue is \$1.298 million and the total cost is \$754,000. This corresponds with the profit of \$545,000, which is the value of the objective function.

In the *DECISION VARIABLE* section you get a list of all the variables in the model, both vector variables and plain variables. In our case, we have a single vector variable *Produce* defined over the index *product*. You will see that for products *A1* and *A3* the solution suggests that you produce 4300 and 5400 units respectively. This is the same amount as the demand for those products. On the other hand, product *A2* is suggested to produce 1611 units which is less than the demand. Clearly we did not have the capacity to produce enough to fulfill the demand for all of the products and the model chose products *A1* and *A3* to fulfill the demand.

By the way, you might have noticed that the value for *A2* in the solution output is actually 1611.8182 units instead of 1612. This results from the fact that all variables in Linear Programming models are by default continuous. In this model it does not matter very much and we can just round it up to the nearest number 1612, but if it did, you could have restricted the variable to take only integer values by specifying it as an integer variable in the model.

In the *CONSTRAINTS* section the solution file lists all of the constraints for the model, again both plain and vector constraints. In our model we had a single plain constraint called *ProdCapacity*. Since the slack for the constraint is zero this means that we are running at full capacity. The shadow price tells you what the marginal cost would be if you needed to reduce the limit of the constraint by one unit. Since the production capacity constraint has production days as the unit, reducing the days available by one day the profit would decrease by \$21,195.

Part IV A MPL Tutorial



SESSION 4:

A Production Planning Model with Multiple Time Periods

You will now expand the model from the previous session to include multiple time periods. A new *period index* is introduced into the model to cover these time periods and you will then update the various vectors that have been affected to account for the new index.

4.1 New Concepts in this Session

Period Indexes

In order to update your model to include a multiple period, you will need to create an index that represents that time period. This type of index is called a *period index*. After you have defined the period index, you can then use it to update data, variable and constraint vectors to include a specified time period. Examples of period indexes might include:

- months
- quarters
- years

Sales and Inventory Variables

When you are upgrading a model from a single period to multiple periods, the sales for a specific period may differ from the amount produced in that same period. As a result, a new variable is needed that represents how much needs to be sold, and a new variable that represents the inventory level for each period.

In many cases, there is a fixed cost per period involved with storing inventory. In other cases, however, the cost might not be connected to the actual storing of the inventory, but rather with the stocking and removing of the products from the inventory storage, through the labor costs involved. This means you would need to add more variables to the model, one for stocking the products, and another for removing them from the inventory.

Inventory Balance Constraints

As it is not possible to sell more of the products than you have on hand, the inventory variable is used to connect the production variables to the sales variables. This is done through a type of constraint, typically called a *balance constraint*. Balance constraints are used to ensure that quantities going into an entity *equal* the quantities going out.

Session 4 Planning Model with Multiple Time Periods

A typical inventory balance constraint, stipulates that the total production, plus the inventory level from the previous period, *equals* the total amount sold, plus how much you leave in the inventory. An example of an inventory balance constraint is:

$$\text{produce} + \text{Inventory}[\text{month}-1] = \text{sales} + \text{Inventory}$$

In this case, the entity being balanced is the inventory. The *Inventory[month-1]* is a notation used in **MPL** to represent the previous period. When you are working with an inventory where a cost needs to be applied to the stocking and removing of the inventory, you will need to define two balance constraints. For example:

$$\text{Produce} + \text{OutInv} = \text{Sales} + \text{PutInv}$$

$$\text{PutInv} + \text{Inventory}[\text{month}-1] = \text{OutInv} + \text{Inventory}$$

If you think of the plant as an entity, then in the first constraint we are balancing what goes into the plant with what goes out. In the same manner, the second constraint balances what goes into and what goes out of the inventory.

Initial and Ending Inventory

In many cases the model developer needs to specify a specific initial or ending inventory for the planning period. **MPL** by default excludes entries like *Inventory[month-1]* for month equal to zero. There are several ways you can specify a starting inventory, for example you can enter the constraint in two parts as follows:

```
INDEX
  month := (Jan, Feb, Mar, Apr)

DATA
  StartInv := 450

SUBJECT TO
  InitInv[month=Jan]:
    produce + StartInv = sales + Inventory

  InvBal[month>Jan]:
    produce + Inventory[month-1] = sales + Inventory
```

This will create a balance constraint for the month of January with starting inventory of 450 units. There are other ways of including a starting inventory in **MPL** that does not require you to duplicate the constraint, for example with subindexes, but this is the simplest way to formulate these kind of constraints.

4.2 Problem Description: A Multi-Period Production Planning Model

In this session, you will create the new model formulation for a *multi-period production planning* model. You will use the model you created in session 3, and make the necessary additions and updates to it.

In this new problem, you have a planning period of four months, from January to April. You need to create an index that contains the four months mentioned, and then update the rest of the model accordingly, by adding the index to the defined vectors.

As in the problem in session 3, the selling price for each product is still \$120.00, \$100.00, and \$115.00, respectively. Now, instead of having a single demand for each product, you have a separate demand for each product *and* each month, as given in the table below:

<i>Product Demand</i>	<i>Jan</i>	<i>Feb</i>	<i>Mar</i>	<i>Apr</i>
<i>A1</i>	4300	4200	6400	5300
<i>A2</i>	4500	5400	6500	7200
<i>A3</i>	5400	6700	7800	8200

The production rate and the production cost remain the same, as in given in the table in session 3. Notice that the production days available are different for each month with 23 days for January, 20 for February, 23 for March, and 22 for April.

You will be introducing inventory into the model, therefore, you have the inventory cost for each product with *A1* - \$3.50/month, *A2* - \$4.00/month, and *A3* - \$3.00/month, respectively.

Each product takes up the same space, but the total inventory capacity is now 800 units per month.

4.3 Formulation of the Model in MPL

Listed below is the entire model formulation for *Planning4*. As you can see the model has expanded somewhat from session 3. The additions to the model are highlighted in boldface in order to make it easy for you to see the changes.

```

TITLE
  Production_Planning4;

INDEX
  product := (A1, A2, A3);
  month   := (Jan, Feb, Mar, Apr);

DATA
  Price[product]           := (120.00, 100.00, 115.00);
  Demand[product, month] := (4300, 4200, 6400, 5300,
                             4500, 5400, 6500, 7200,
                             5400, 6700, 7800, 8200);

  ProdCost[product]       := (73.30, 52.90, 65.40);
  ProdRate[product]       := (500, 450, 550);
  ProdDaysAvail[month]    := (23, 20, 23, 22);
  InvCost[product]       := (3.50, 4.00, 3.00);
  InvCapacity           := 800;

VARIABLES
  Produce[product, month] -> Prod;
  Inventory[product, month] -> Inv;
  Sales[product, month]    -> Sale;

MACROS
  TotalRevenue := SUM(product, month: Price * Sales);
  TotalProdCost := SUM(product, month: ProdCost * Produce);
  TotalInvCost  := SUM(product, month: InvCost * Inventory);
  TotalCost    := TotalProdCost + TotalInvCost;

MODEL

  MAX Profit = TotalRevenue - TotalCost;

SUBJECT TO
  ProdCapacity[month] -> PCap:
    SUM(product: Produce / ProdRate) <= ProdDaysAvail;

  InvBal[product, month] -> IBal:
    Produce + Inventory[month-1] = Sales + Inventory;

  MaxInventory[month] -> MaxI:
    SUM(product: Inventory) <= InvCapacity;

BOUNDS
  Sales <= Demand;

END

```

4.4 Enter New Elements to the Model Step-by-Step

Step 1: Start MPL and Create a New Model

1. Start the **MPL** application.
2. Choose *File / Open* and open the model from the previous session *Planning3.mpl*.
3. Choose *File / Save As* to save it as a new model file *Planning4.mpl*.

Step 2: Change the Title for the Model

Change the title for the model to reflect that you are working with the *Planning4* model:

```
TITLE
  Production_Planning4;
```

Step 3: Add the Index 'month' to the Model

In this example, there is a planning period of four months represented by an index called *month*. This index will have four elements *Jan*, *Feb*, *Mar*, and *Apr*, to represent the four month planning period. Add the following definition for the *month* index, that is shown in boldface, to the *INDEX* section:

```
INDEX
  product := (A1, A2, A3);
  month   := (Jan, Feb, Mar, Apr);
```


Step 4: Update the 'Demand' Data Vector to Include the 'month' Index

In the *DATA* section, most of the data definitions are the same as in session 3. One of the data vectors; *Demand*, needs to be upgraded to include the *month* index, as it now has different data values for each month. The values for this vector are given in the table in the problem description earlier in this session. Add the index *month* to the declaration for the *Demand* data vector, followed with a list of data values as shown below:

```
DATA
Price[product]      := (120.00, 100.00, 115.00);
Demand[product, month] := (4300, 4200, 6400, 5300,
                           4500, 5400, 6500, 7200,
                           5400, 6700, 7800, 8200);
```

Step 5: Update the 'ProdDaysAvail' Data Constant to a Data Vector over the Index 'month'

The data constant *ProdDaysAvail* now has a different value for each month, as does the *Demand* data vector. This means it needs to be upgraded from a data constant to a one dimensional data vector, with the *month* as the index. Using the list of production days available, found in the problem description earlier in this session, update the *ProdDaysAvail* as follows:

```
ProdCost[product]      := (73.30, 52.90, 65.40);
ProdRate[product]      := (500, 450, 550);
ProdDaysAvail[month]   := (23, 20, 23, 22);
```

Step 6: Add Data Vectors for Inventory Cost and Inventory Capacity

The problem description defined a cost for each product stored in inventory and a limit on how much can be stored in the inventory. Therefore, in order to represent this, add one more data vector to the model; *InvCost*, and also a new data constant, *InvCapacity*. At the end of the *DATA* section add the following definitions:

```
InvCost[product]      := (3.50, 4.00, 3.00);
InvCapacity           := 800;
```

Step 7: Add Inventory and Sales Variables to the Model

With this model there are two new variables, *Sales* and *Inventory*, that need to be introduced into the model. The *Sales* variable is used to represent how much of each product is sold each month. The *Inventory* variable is used to represent how much of each product is stored, each month. The *Produce* variable needs to be upgraded to include the index *month* as different amounts of each product are produced, each month. In the model add the following definitions to the *VARIABLES* section:

```
VARIABLES
  Produce[product, month]  -> Prod;
  Inventory[product, month] -> Inv;
  Sales[product, month]    -> Sale;
```

As in the previous model, the name that appears after the ‘->’ (read becomes) sign is an optional abbreviation of the vector name. This is used to offset the variable name size limitations of many LP solvers.

Step 8: Add the Inventory Cost to the Objective Function

In the model from the previous session, the total revenue and the total production cost were included in the objective function. Now you need to update this objective function with the index; *month*, and add an entry for the total inventory cost. As in the previous session, you will continue to use macros to represent each summation.

When calculating the total revenue, you will now use the *Sales* variable instead of the *Produce* variable and add the index *month* to the summation. For the total production cost, you will also need to upgrade the summation to include the *month* index. The total inventory cost will be defined, as the inventory cost, times the inventory level, for each *product* and *month*.

To make changes in the *MACROS* section, replace the *Produce* variable with the *Sales* variable, update the total revenue and the total production cost summations to include the *month* index, and add a new macro definition for the total inventory cost as follows:

```
MACROS
  TotalRevenue := SUM(product, month: Price * Sales);
  TotalProdCost := SUM(product, month: ProdCost * Produce);
  TotalInvCost := SUM(product, month: InvCost * Inventory);
  TotalCost := TotalProdCost + TotalInvCost;
```

Please note that the macro for the total production cost has been renamed to *TotalProdCost*. Another new macro; *TotalCost*, has been added where you can sum together these two macros to get the total cost.

Session 4 Planning Model with Multiple Time Periods

This allows the objective function definition to remain unchanged:

```
MODEL
    MAX Profit = TotalRevenue - TotalCost;
```

Step 9: Update the Production Capacity Constraint for Multiple Months

In the *production capacity* constraint, add the index *month* to the constraint definition and the rest of the constraint remains the same.

```
SUBJECT TO
    ProdCapacity[month] -> PCap:
        SUM(product: Produce / ProdRate) <= ProdDaysAvail;
```

Please note that in **MPL** you do not have to enter each index subscript when referring to the data and variable vectors. This means you can easily add more indexes to constraints without having to change how you refer to each vector.

Step 10: Add an Inventory Balance Constraint to the Model

The addition of the *Inventory* variable to the model needs to include a standard inventory balance constraint. This constraint ranges over each product and each month, specifying that the production, plus the inventory from the month before, is *equal* to the amount sold, plus the inventory for the current month. Add the following *InvBal* constraint below the previous *ProdCapacity* constraint:

```
InvBal[product, month] -> IBal:
    Produce + Inventory[month-1] = Sales + Inventory;
```

When entering previous time periods, as in this case, the month before, **MPL** allows you to use expressions such as *[month-1]*.

Step 11: Add an Inventory Capacity Constraint to the Model

There is a limit on how much inventory space is available. Therefore, you need to add an inventory capacity constraint to the model. In the problem description, you were given that each product takes up an equal amount of space in inventory and that you can add, or sum over, all of the products to get the total inventory space used. Add the following constraint definition to the model:

```
MaxInventory[month] -> MaxI:  
SUM(product: Inventory) <= InvtCapacity;
```

Step 12: Update the Maximum Demand Upper Bound to use the 'Sales' Variable

In the maximum demand upper bound you need to update it to include the *Sales* variable instead of the *Produce* variable as shown below:

```
BOUNDS  
Sales <= Demand;
```

After you have finished entering the model, you should save it by choosing *Save* from the *File* menu.

4.5 Solve the Model and Analyze the Solution

The next step is to solve the *Planning4* model, by choosing *Solve CPLEX* from the *Run* menu. If you have entered the data correctly, **MPL** will display the message *Optimal Solution Found*. If there is an error message window, with a syntax error, please check the formulation you entered with the model detailed earlier in this session.

After solving the model **MPL**, automatically creates a standard solution file named *Planning4.sol*. You can display the solution file in a view window by pressing the *View* button at the bottom of the *Status Window*. A full listing of the solution file is shown below:

```

MPL Modeling System - Copyright (c) 1988-2001, Maximal Software, Inc.
-----
MODEL STATISTICS

Problem name:      Production_Planning4

Filename:         Planning4.mpl
Date:            April 18, 1998
Time:            22:52
Parsing time:    0.15 sec

Solver:          CPLEX
Objective value: 2246007.27273
Iterations:      26
Solution time:   0.04 sec

Constraints:      20
Variables:       36
Nonzeros:        69
Density:         10 %

SOLUTION RESULT

Optimal solution found

MAX Profit = 2246007.2727

MACROS

Macro Name          Values
-----
TotalRevenue        5386045.4545
TotalProdCost       3139078.1818
TotalInvtCost        960.0000
TotalCost           3140038.1818
-----

```

Part IV A MPL Tutorial

DECISION VARIABLES

VARIABLE Produce[product,month] :

product	month	Activity	Reduced Cost
A1	Jan	4300.0000	0.0000
A1	Feb	4200.0000	0.0000
A1	Mar	4409.0909	0.0000
A1	Apr	3545.4545	0.0000
A2	Jan	1800.0000	0.0000
A2	Feb	0.0000	-3.6667
A2	Mar	0.0000	-4.7889
A2	Apr	0.0000	-0.7889
A3	Jan	5720.0000	0.0000
A3	Feb	6380.0000	0.0000
A3	Mar	7800.0000	0.0000
A3	Apr	8200.0000	0.0000

VARIABLE Inventory[product,month] :

product	month	Activity	Reduced Cost
A1	Jan	0.0000	-0.2000
A1	Feb	0.0000	-2.4900
A1	Mar	0.0000	-3.5000
A1	Apr	0.0000	-123.5000
A2	Jan	0.0000	-4.0000
A2	Feb	0.0000	-4.0000
A2	Mar	0.0000	0.0000
A2	Apr	0.0000	-108.0000
A3	Jan	320.0000	0.0000
A3	Feb	0.0000	-2.0818
A3	Mar	0.0000	-3.0000
A3	Apr	0.0000	-110.8545

VARIABLE Sales[product,month] :

product	month	Activity	Reduced Cost
A1	Jan	4300.0000	4.3100
A1	Feb	4200.0000	1.0100
A1	Mar	4409.0909	0.0000
A1	Apr	3545.4545	0.0000
A2	Jan	1800.0000	0.0000
A2	Feb	0.0000	0.0000
A2	Mar	0.0000	0.0000
A2	Apr	0.0000	-4.0000
A3	Jan	5400.0000	11.0636
A3	Feb	6700.0000	8.0636
A3	Mar	7800.0000	7.1455
A3	Apr	8200.0000	7.1455

Session 4 Planning Model with Multiple Time Periods

CONSTRAINTS

CONSTRAINT ProdCapacity[month] :

month	Slack	Shadow Price
Jan	0.0000	-21195.0000
Feb	0.0000	-22845.0000
Mar	0.0000	-23350.0000
Apr	0.0000	-23350.0000

CONSTRAINT InvtBal[product,month] :

product	month	Slack	Shadow Price
A1	Jan	0.0000	115.6900
A1	Feb	0.0000	118.9900
A1	Mar	0.0000	120.0000
A1	Apr	0.0000	120.0000
A2	Jan	0.0000	100.0000
A2	Feb	0.0000	100.0000
A2	Mar	0.0000	100.0000
A2	Apr	0.0000	104.0000
A3	Jan	0.0000	103.9364
A3	Feb	0.0000	106.9364
A3	Mar	0.0000	107.8545
A3	Apr	0.0000	107.8545

CONSTRAINT MaxInventory[month] :

month	Slack	Shadow Price
Jan	480.0000	0.0000
Feb	800.0000	0.0000
Mar	800.0000	0.0000
Apr	800.0000	0.0000

END

Part IV A MPL Tutorial

According to the solution, the profit is now \$2.2M which is considerably higher than in the *Planning3* model, as we are now working with four months. This comes from a total revenue of \$5.4M and a total cost of \$3.1M, most of which is the production cost, as we keep very low inventory just for January.

If you look at the *Produce* variable in the solution, you will notice that we are producing products *A1* and *A3* for the whole planning period, although not always up to the full demand. Product *A2*, on the other hand, only produced 1800 units in *January*, as we do not have enough capacity to produce all three of the products.

In January, the model decided to produce an extra 320 units *A3*, above the required demand, in order to put enough into inventory satisfy the demand in February.

SESSION 5:

A Production Planning Model with Multiple Plants

When formulating a model for larger companies, you will often encounter models that are not limited to a single plant. In this session, you will create a *production planning* model that includes multiple plants. You will take the model from the previous session and upgrade it to include another index, *plants*, which will represent all of the plants that are available in order to produce the products. You will then go through the model, step by step, and update all the variable vectors and constraints to account for the new index.

5.1 New Concepts in this Session

Plants and other Location Indexes

Location indexes are quite common when formulating production planning models. One example of a location index, would be to represent the plants where the company produces the products for the company. Other examples would include warehouses, factories, distribution centers, etc.

It is common, when working with models that include location indexes, that shipping is allowed between the locations. These models are often called *transportation* or *distribution* models and will be covered in later sessions.

External Data Files

When formulating small models it is acceptable to leave the data definitions inside the model. As soon as you start working with multi-dimensional models, this becomes difficult to manage, and it is necessary to move the data into separate data files. Keeping the data separate from the model, enhances the readability of the model and make the data easier to maintain.

The model you are creating in this session has multiple indexes, *product*, *month*, and *plant*, and data vectors such as *Demand*, that are two-dimensional and could be moved into a separate data file. In the model, instead of listing all the data elements for the data vector, you can use the keyword *DATAFILE* and then the name of the data file as shown here below:

```
demand[product, month] := DATAFILE("Demand.dat");
```

5.2 Problem Description: A Production Planning Model with Multiple Plants

In this session, you will create a new model formulation for the *production planning* model to include multiple plants as well as the multiple periods that were introduced in session 4. What you want to decide is how much to produce of each product, for each month, in each plant, as well as how much to sell and store in inventory, for each month, in each plant.

In this new problem you are going to have four different plants *p1*, *p2*, *p3*, and *p4*. Any of these plants can produce all three of the products. Create an index called *plants* that contains the four different plants and then update the model accordingly by adding the index to the applicable vectors.

As in the previous session, the selling price stays the same for each product, \$120.00, \$100.00, and \$150.00, respectively. The product demands also remain the same as in the previous session. Refer to the demand table in session 3 for the necessary data.

Now that you have multiple plants the production cost for each product is different for each plant. This data is in the table below.

<i>Production Cost</i>	<i>A1</i>	<i>A2</i>	<i>A3</i>
<i>plant 1</i>	\$73.30	\$52.90	\$65.40
<i>plant 2</i>	\$79.00	\$52.00	\$66.80
<i>plant 3</i>	\$75.80	\$52.10	\$50.90
<i>plant 4</i>	\$82.70	\$63.30	\$53.80

The production rate for each product is also different for each plant as shown in the table below:

<i>Production Rate</i>	<i>A1</i>	<i>A2</i>	<i>A3</i>
<i>plant 1</i>	500	450	450
<i>plant 2</i>	550	450	300
<i>plant 3</i>	450	350	300
<i>plant 4</i>	550	400	350

5.3 Formulation of the Model in MPL

Listed below is the entire model formulation for *Planning5*. The additions to the model are highlighted in boldface in order to make it easy for you to see the changes from the model in session 4.

```
TITLE
  Production Planning5;

INDEX
  product := (A1, A2, A3);
  month   := (Jan, Feb, Mar, Apr);
  plant   := (p1, p2, p3, p4);

DATA
  Price[product]      := (120.00, 100.00, 115.00);
  Demand[product, month] := DATAFILE("Demand.dat");
  ProdCost[plant, product] := DATAFILE("ProdCost.dat");
  ProdRate[plant, product] := DATAFILE("ProdRate.dat");
  ProdDaysAvail[month] := (23, 20, 23, 22);
  InvtCost[product]    := (3.50, 4.00, 3.00);
  InvtCapacity         := 800;

VARIABLES
  Produce[plant, product, month] -> Prod;
  Inventory[product, month]       -> Invt;
  Sales[product, month]           -> Sale;

MACROS
  TotalRevenue := SUM(product, month: Price * Sales);
  TotalProdCost := SUM(plant, product, month: ProdCost * Produce);
  TotalInvtCost := SUM(product, month: InvtCost * Inventory);
  TotalCost     := TotalProdCost + TotalInvtCost;

MODEL

  MAX Profit = TotalRevenue - TotalCost;

SUBJECT TO
  ProdCapacity[plant, month] -> PCap:
    SUM(product: Produce / ProdRate) <= ProdDaysAvail;

  InvtBal[product, month] -> IBal:
    SUM(plant: Produce) + Inventory[month-1] = Sales + Inventory;

  MaxInventory[month] -> MaxI:
    SUM(product: Inventory) <= InvtCapacity;

BOUNDS
  Sales <= Demand;

END
```

5.4 Enter New Elements to the Model Step-by-Step

Step 1: Start MPL and Create a New Model

1. Start the **MPL** application.
2. Choose *File / Open* and open the model from the previous session *Planning4.mpl*.
3. Choose *File / Save As* to save it as a new model file *Planning5.mpl*.

Step 2: Change the Title for the Model

Change the title for the model to reflect that you are working with the *Planning5* model:

```
TITLE
  Production_Planning5;
```

Step 3: Add the Location Index 'plant' to the Model

In this model, you have four different plant locations which you will represent by creating. Create a new index which you will call *plant*. This index will have four elements *p1*, *p2*, *p3*, and *p4* to represent each period of the four month planning period. Add the following definition for the *plant* index to the *INDEX* section:

```
INDEX
  product := (A1, A2, A3);
  month   := (Jan, Feb, Mar, Apr);
  plant   := (p1, p2, p3, p4);
```

Step 4: Change the definition of the 'Demand' Data Vector to Read Data from an External Data File

In this session, you are going to move the data values for the two-dimensional data vectors to external data files. When working with data vectors that have two dimensions or higher, it is often a good idea to move the data values to an external data file instead of listing all of the numbers directly in the model file. This keeps the data separate from the model, enhances the readability of the model, and makes the data easier to maintain.

The first data vector you want to move to an external data file is the *Demand* data vector. In the *DATA* section, use the *Cut* command from the *Edit* menu to remove the list of numbers for the *Demand* vector and then enter the keyword *DATAFILE* and the filename *Demand.dat* in its place as follows:

```
DATA
Price[product]          := (120.00, 100.00, 115.00);
Demand[product, month] := DATAFILE("Demand.dat");
```

Step 5: Create the Data File Demand.dat

The next step is to create the data file *Demand.dat*. First, open a new model editor window by choosing *New* in the *File* menu. If you used the *Edit | Cut* command in the previous *Step 4* to remove the data values, they are now in the *Clipboard* and you can use the *Edit | Paste* command to place the data back into the data file. Otherwise, you can use the demand data table from the problem description in session 3 to enter the data values into the data file as follows:

```
! Demand.dat - Demand per month for each product
!
! Demand[product,month]:
!
!      Jan    Feb    Mar    Apr
!      -----
!      4300,  4200,  6400,  5300,
!      4500,  5400,  6500,  7200,
!      5400,  6700,  7800,  8200
```

The lines that start with exclamation marks are comments used to enhance the readability. The numbers in the data file can be separated by comma or space or both. After you have entered all of the data save the file as *Demand.dat* in the *Tutorial* folder.

Step 6: Upgrade the 'ProdCost' and 'ProdRate' Data Vectors to Include the 'plant' Index

Two of the data vectors, *ProdCost* and *ProdRate*, need to be upgraded to include the *plant* index. The *ProdCost* data vector is now defined over two domain indexes, *plant* and *product*, and will given data values from an external data file. The *ProdRate* data vector will also be given values from a data file. In the model editor, add the index *plant*, to the declaration of both the *ProdCost* and the *ProdRate* data vectors, and follow it with the data filenames *ProdCost.dat* and *ProdRate.dat* respectively as follows:

```
ProdCost[plant, product] := DATAFILE("ProdCost.dat");
ProdRate[plant, product] := DATAFILE("ProdRate.dat");
ProdDaysAvail[month]     := (23, 20, 23, 22);
InvCost[product]         := (3.50, 4.00 3.00);
InvCapacity              := 800;
```

Step 7: Create the Data Files for the 'ProdCost' and 'ProdRate' Data Vectors

Now open a new editor window by selecting *File / New* in the menu to enter the data file. Type in the data from the *Production Cost* table in the problem description as follows:

```
!
! ProdCost.dat - Cost per item produced
!
! ProdCost[plant, product]:
!
!      A1      A2      A3
!      -----
!      73.30,  52.90,  65.40,
!      79.00,  52.00,  66.80,
!      75.80,  52.10,  50.90,
!      82.70,  63.30,  53.80
```

Again, the lines that contain exclamation marks are comments used to enhance the readability. After you have entered all of the data save the file using the name *ProdCost.dat*.

Part IV A MPL Tutorial

For the production rate create a new data file called *ProdRate.dat* using the values from the table in the problem description.

```
!
! ProdRate.dat - Items produced per day
!
! ProdRate[plant, product]:
!
!      A1    A2    A3
!      -----
!      500,  450,  450,
!      550,  450,  300,
!      450,  350,  300,
!      550,  400,  350
```

Step 8: Update the 'Produce' Variable to Include the 'plant' Index

In order to determine how much you want to produce of each product, for each plant, that you need to add the *plant* index to the vector definition of the *Produce* variable as follows:

```
VARIABLES
  Produce[plant, product, month] -> Prod;
  Inventory[product, month]      -> Invt;
  Sales[product, month]          -> Sale;
```

Step 9: Add the 'plant' Index to the 'TotalProdCost' Summation

Since the *Produce* vector variable now includes the new index, *plant*, the calculation of the total production cost in the *MACROS* section needs also to be updated to include the *plant* index:

```
MACROS
  TotalRevenue := SUM(product, month: Price * Sales);
  TotalProdCost := SUM(plant, product, month: ProdCost * Produce) ;
  TotalInvtCost := SUM(product, month: InvtCost * Inventory);
  TotalCost := TotalProdCost + TotalInvtCost;
```

The objective function itself does not change as you are using the same macros as in the previous session.

Step 10: Add the 'plant' Index to the 'ProdCapacity' Constraint

The change for the production capacity constraint is very simple. Add the index *plant* to the production capacity declaration and the rest of the constraint remains the same.

```
SUBJECT TO
  ProdCapacity[plant, month] -> PCap:
    SUM(product: ProdRate / Produce) <= ProdDaysAvail;
```

Step 11: Add Summation of the 'Produce' Variable Over the 'plant' Index to the Inventory Balance Constraint

You can now produce the products in any of the four plants, therefore, you need to update the inventory balance constraint to include a summation, over all of the plants, of the *Produce* variable vector.

```
InvBal[product, month] -> IBal:
  SUM(plant: Produce) + Inventory[month-1] = Sales + Inventory;
```

After you have finished entering the model, you should save it by choosing *Save* from the *File* menu.

5.5 Solve the Model and Analyze the Solution

Since we have added more indexes to the model, the number of variables has increased considerably. Typically, when working with larger models, the model developer wants to include only the variables that have nonzero values. **MPL** has a number of options dialog boxes in the *Options* menu where you can change the default behavior of the program. One of the dialog boxes is the *Solution File Options* Dialog Box where you can adjust what is included in the solution file. To change the default to include nonzero values in the solution file only, do the following:

1. From the *Options* menu choose *Solution File* to open the *Options Dialog Box* shown here below:

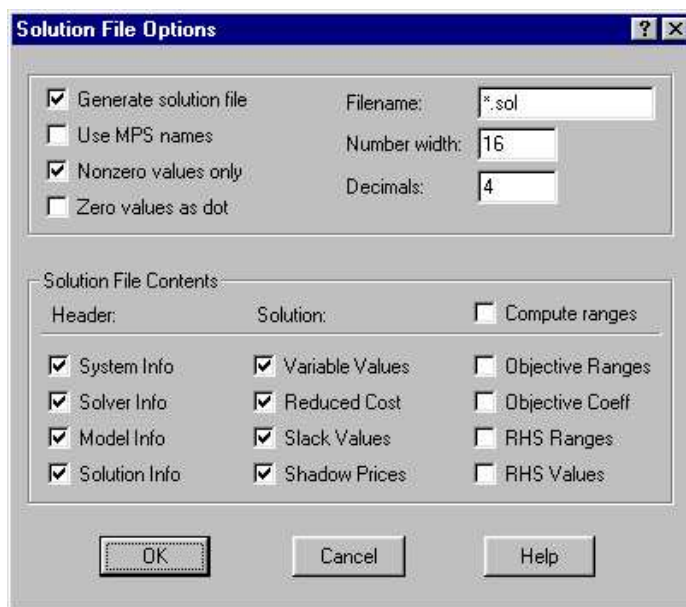


Figure T5.1: The Solution File Options Dialog Box

2. Turn the *Nonzero Values Only* check box *On* by clicking on it.
3. Close the dialog box by pressing the *OK* button.

After changing the *Nonzero values only* option, the next step is to solve the model by choosing *Solve CPLEX* from the *Run* menu. If everything goes well **MPL** will display the message “*Optimal Solution Found*”. If there is an error message window with a syntax error please check the formulation you entered with the model listing detailed earlier in this session.

As the models you are working with become bigger you tend to look at only certain parts of the solution instead of the whole solution file. This time, instead of listing the whole solution file, we are going to use the model definitions tree window to view only the parts of the solution we are interested in.

The Model Definitions Window allows you to see all of the defined items from the model formulation in a hierarchical tree where each branch corresponds to a section in the model. While in **MPL** it is normally a good idea to leave the tree window open at all times. **MPL** will then automatically update its contents every time you solve your model. To look at the Model Definitions Window for the *Planning5* model choose *Model Definitions* from the *View* menu.

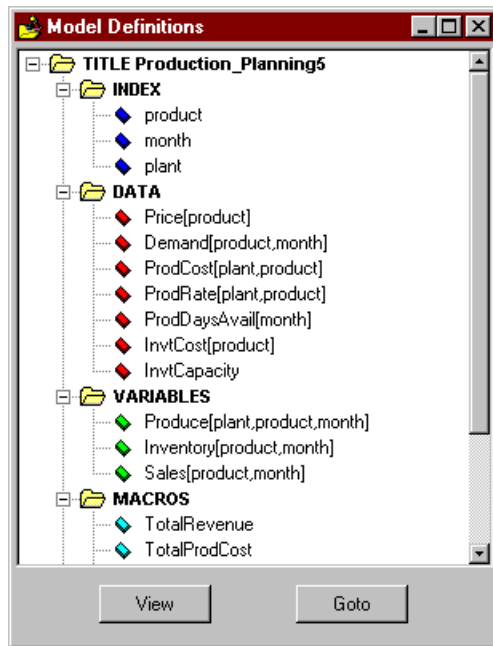


Figure T5.2: The Model Definitions Window for Planning5

From the tree window you can select any of the defined items in the model to look at the actual values for that item. For example, to look at the values for the *Produce* variable, either double click on the *Produce* item in the tree, or alternatively select it and then press the *View* button. This will display a view window containing only the values for the *Produce* variable.

Part IV A MPL Tutorial

VARIABLE Produce[plant,product,month] :

plant	product	month	Activity	Reduced Cost
p1	A1	Jan	4300.0000	0.0000
p1	A1	Feb	4200.0000	0.0000
p1	A1	Mar	6400.0000	0.0000
p1	A1	Apr	5300.0000	0.0000
p2	A2	Jan	4500.0000	0.0000
p2	A2	Feb	5400.0000	0.0000
p2	A2	Mar	6500.0000	0.0000
p2	A2	Apr	7200.0000	0.0000
p3	A3	Jan	5400.0000	0.0000
p3	A3	Feb	6000.0000	0.0000
p3	A3	Mar	6900.0000	0.0000
p3	A3	Apr	6600.0000	0.0000
p4	A3	Feb	700.0000	0.0000
p4	A3	Mar	900.0000	0.0000
p4	A3	Apr	1600.0000	0.0000

If you look at the activity values for the *Produce* variable you will see that this time we are fulfilling the demand for all the products since we now have enough capacity. The model decides which plants are used for which products. For example, plant *p1* is used to produce product *A1*, plant *p2* is used to produce product *A2*, and plants *p3* and *p4* are used to produce product *A3*.

If you go to the tree window again and open up a window for the *ProdCapacity* constraint you will get the following solution values.

CONSTRAINT ProdCapacity[plant,month] :

plant	month	Slack	Shadow Price
p1	Jan	14.4000	0.0000
p1	Feb	11.6000	0.0000
p1	Mar	10.2000	0.0000
p1	Apr	11.4000	0.0000
p2	Jan	13.0000	0.0000
p2	Feb	8.0000	0.0000
p2	Mar	8.5556	0.0000
p2	Apr	6.0000	0.0000
p3	Jan	5.0000	0.0000
p4	Jan	23.0000	0.0000
p4	Feb	18.0000	0.0000
p4	Mar	20.4286	0.0000
p4	Apr	17.4286	0.0000

There is a great deal of slack for each plant and month in the production capacity constraint. This can be interpreted to mean that we could produce a lot more of the products, but it is not necessary as we are already fulfilling the demand. Since the production capacity constraint uses production days as the unit of measure, the slack values represent how many days per month each plant is idle.

SESSION 6:

Upgrade the Model to Allow

Shipments Between the Plants

In session 5, you encountered a model for a production company that used multiple plants to produce their product. What you should note about that model was that while each of the different plants could be used to produce the products individually, all the selling and inventory was handled collectively, as a single source, for the whole company. You will now upgrade that model to allow each plant to sell the products, and maintain inventory individually. Furthermore, in order to fulfill the demand in the most efficient manner, the products can be shipped between the plants as needed.

To upgrade the model you will include the two new alias indexes, *toplant*, and *fromplant*, which will represent the locations you will be shipping to and from. You will also update the *Inventory* and *Sales* variables to include the *plant* index, as each plant can now sell the products and maintain inventory independent of each other.

6.1 New Concepts in this Session

Transportation Models

Models that allow shipping between locations are sometimes called *transportation* or *distribution* models. Typically, in transportation models, you have sources with certain availability, destinations with certain requirements, and you need to ship the products from the sources to the destinations. In some cases, you have transportation models with multiple levels. For example, there can be shipments from plants to depots, and then from the depots to retail stores.

Transshipment Models

Another group of distribution models is *transshipment* models. These models typically arise when you have multiple locations that both produce the goods and also act as demand centers. Since there are no specific sources and destinations, you can ship from any one location to any of the other locations.

Alias Indexes

Alias indexes are useful when you need to define a vector, which uses the same index more than once, as a subscript. When shipping products between plants you need to create a variable vector, representing how much to ship between the plants. Since the source plants and the destination plants come from the same set of plants, you need two alias indexes for the plants. The first alias index is needed to represent the source plants, and the second to represent the destination plants.

Using Where Conditions on Vector Variables

Sometimes, when working with multi-dimensional vector variables you will encounter cases where not all elements of the vector are valid or have a meaning. For example, in transshipment models it would make no sense to ship a product from a certain plant back to that same plant. In these cases, you can use a *WHERE* condition on the variable to remove unnecessary elements.

Session 6 Allow Shipments Between Plants

For example, in a transshipment model you can eliminate the possibility of shipping to the same location by defining the variable as follows:

```
VARIABLES
  Ship[fromplant,toplant]
  WHERE (fromplant <> toplant);
```

In this case, the condition (*fromplant <> toplant*) removes all of the vector elements where the source plant is the same as the destination plant.

In some cases, elements need to be excluded that are not based on the values of the indexes. They then must be based on some data vector in the model. Typically, you have a cost vector assigned to the shipping containing how much it costs to ship between the plants. For those plants, if shipping between is not feasible, you can enter a special value for the cost, such as a zero, to be used to identify them. Then, in the variable definition, you can use this data vector to exclude the shipping routes that are not feasible as follows:

```
VARIABLES
  Ship[fromplant,toplant]
  WHERE (ShipCost[fromplant,toplant] > 0);
```

Plant Balance Constraints

When working with transshipment models you need to ensure that the amount of products shipped to a plant plus how much is produced and pulled from inventory is *equal* to how much is shipped from the plant plus how much is sold and put back into inventory. In short, everything that goes into the plant must be equal to everything that goes out of the plant. This kind of constraint is typically called a plant balance constraint. Here is an example of a simple plant balance constraint:

```
PlantBal[plant, product, month]:
  Produce + Inventory[month-1]
  + SUM(fromplant: Ship[fromplant, toplant:=plant])
  =
  Sales + Inventory
  + SUM(toplant: Ship[fromplant:=plant, toplant]);
```

You will notice that this constraint is similar to the inventory balance constraint you encountered in previous sessions. The only difference is that now you have to take into account that we are shipping to and from each plant by entering a summation over each plant for the *Ship* variable.

The index assignment '*toplant:=plant*', in the first summation, allows us to specify that the *toplant* subscript should take the value of the plant subscript for the *PlantBal* constraint. This summation adds together all the shipments from each of the plants to the particular plant in that constraint. In similar manner, the index assignment '*fromplant:=plant*', in the second summation, specifies that the *fromplant* subscript should take the value of the plant subscript.

6.2 Problem Description: Additions to Allow Shipments Between Plants

In this session, a new model will be created where each plant now acts as a separate demand center for the products and can also keep inventory. You will use the model you created in session 5 and make the necessary additions and updates to it.

Since each plant can sell the products, we now have a different demand for each plant, as well as for each product, and each month. The demand is given in the table below:

Demand Table

<i>Plant</i>	<i>Product</i>	<i>Jan</i>	<i>Feb</i>	<i>Mar</i>	<i>Apr</i>
<i>p1</i>	<i>A1</i>	4300	4200	6400	5300
	<i>A2</i>	4500	5400	6500	7200
	<i>A3</i>	5400	6700	7800	8200
<i>p2</i>	<i>A18</i>	5100	6200	5400	7600
	<i>A2</i>	6300	7100	5200	6300
	<i>A3</i>	4800	6500	5000	7200
<i>p3</i>	<i>A1</i>	4100	6100	4700	5800
	<i>A2</i>	5300	5200	5700	4100
	<i>A3</i>	4200	4100	5200	6300
<i>p4</i>	<i>A1</i>	4300	4100	5300	4500
	<i>A2</i>	5300	6400	4200	6200
	<i>A3</i>	5600	5200	3800	4100

Session 6 Allow Shipments Between Plants

This data has three dimensions, plants, products, and months. In linear programming models it is quite typical to have data with multiple dimensions, possibly up to eight or more. In the next session, we will update the demand data to include one more dimension; machines, creating a four dimensional vector.

The inventory capacity is now different for each plant. We have four capacity values, one for each plant, 800, 400, 500 and 400 respectively.

Since we now have multiple plants, each of which can maintain inventory, we now have different inventory costs for each plant and each product. The new cost values for the inventory are shown here below:

<i>Inventory Cost</i>	<i>A1</i>	<i>A2</i>	<i>A3</i>
<i>p1</i>	\$8.50	\$7.00	\$6.50
<i>p2</i>	\$9.80	\$9.80	\$9.80
<i>p3</i>	\$7.50	\$7.50	\$7.50
<i>p4</i>	\$9.30	\$8.00	\$6.50

Finally, since we are allowing shipments between the plants there are certain costs involved for shipping a product, as shown in the table below:

<i>Shipping Cost</i>	<i>p1</i>	<i>p2</i>	<i>p3</i>	<i>p4</i>
<i>p1</i>	-	\$15.00	\$21.00	\$13.00
<i>p2</i>	\$16.00	-	\$12.00	\$12.00
<i>p3</i>	\$14.00	\$17.00	-	\$15.00
<i>p4</i>	\$21.00	\$13.00	\$10.00	-

As you can see there are no values in the table where the source plant is same as the destination plant as there is no benefit in shipping back to the same plant.

6.3 Formulation of the Model in MPL

The following is the entire model formulation for *Planning6*. As you can see the model has expanded somewhat from session 5.

```

TITLE
  Production_Planning6;
INDEX
  product   := (A1, A2, A3);
  month     := (Jan, Feb, Mar, Apr);
  plant     := (p1, p2, p3, p4);
  fromplant := plant;
  toplant   := plant;

DATA
  Price[product]           := (120.00, 100.00, 115.00);
  Demand[plant, product, month] := DATAFILE("Demand6.dat");
  ProdCost[plant, product] := DATAFILE("ProdCost.dat");
  ProdRate[plant, product] := DATAFILE("ProdRate.dat");
  ProdDaysAvail[month]     := (23, 20, 23, 22);
  InvtCost[plant, product] := DATAFILE("InvtCost.dat");
  InvtCapacity[plant]      := (800, 400, 500, 400);
  ShipCost[fromplant, toplant] := DATAFILE("ShipCost.dat");

VARIABLES
  Produce[plant, product, month] -> Prod;
  Inventory[plant, product, month] -> Invt;
  Sales[plant, product, month] -> Sale;
  Ship[product, month, fromplant, toplant]
    WHERE (fromplant <> toplant);

MACROS
  TotalRevenue := SUM(plant, product, month: Price * Sales);
  TotalProdCost := SUM(plant, product, month: ProdCost * Produce);
  TotalInvtCost := SUM(plant, product, month: InvtCost * Inventory);
  TotalShipCost := SUM(product, month, fromplant, toplant: ShipCost * Ship);
  TotalCost := TotalProdCost + TotalInvtCost + TotalShipCost;

MODEL
  MAX Profit = TotalRevenue - TotalCost;

SUBJECT TO
  ProdCapacity[plant, month] -> PCap:
    SUM(product: Produce / ProdRate) <= ProdDaysAvail;

  PlantBal[plant, product, month] -> PBal:
    Produce + Inventory[month-1]
    + SUM(fromplant: Ship[fromplant, toplant:=plant])
    =
    Sales + Inventory
    + SUM(toplant: Ship[fromplant:=plant, toplant]);

  MaxInventory[plant, month] -> MaxI:
    SUM(product: Inventory) <= InvtCapacity;

BOUNDS
  Sales < Demand ;

END

```

6.4 Enter New Elements to the Model Step-by-Step

Step 1: Start MPL and Create a New Model

1. Start the **MPL** application.
2. Choose *File / Open* and open the model from the previous session *Planning5.mpl*.
3. Choose *File / Save As* to save it as a new model file *Planning6.mpl*.

Step 2: Change the Title for the Model

Change the title for the model to reflect that you are working with the *Planning6* model:

```
TITLE
  Production_Planning6;
```

Step 3: Add Alias Indexes for the Source and Destination Plants to the Model

In this session, you are going to expand the model to allow shipments between the plants. *Alias indexes* are useful when you need to define a vector, which refers to the same index more than once, as a subscript. Alias indexes are an exact copy of a previously defined index.

In this case, you are creating a vector variable representing how much to ship between the plants. Which means you need two alias indexes to represent the source, and destination plants. Add the following definitions for two new alias indexes that you will call *fromplant* and *toplant* at the end of the *INDEX* section:

```
INDEX
  product   := (A1, A2, A3);
  month     := (Jan, Feb, Mar, Apr);
  plant     := (p1, p2, p3, p4);
  fromplant := plant;
  toplant   := plant;
```

Step 4: Update the 'Demand' Data Vector to Include the 'plant' Index

We now have different demand for each plant the definition for the *Demand* data vector needs to be upgraded to include the *plant* index. In the model editor, add the index *plant* to declaration of the *Demand* data vector and change the file name to *Demand6.dat*.

```
DATA
  Price[product]           := (120.00, 100.00, 115.00);
  Demand[plant, product, month] := DATAFILE("Demand6.dat");
```

Since the *Demand* vector has now three indexes the data file for it needs to be updated. Therefore, you are creating a new data file called *Demand6.dat* using the data values from the table in the problem description given earlier in this session. Enter the data values to the data file as follows:

```
!
! Demand6.dat - Demand for each product and each plant
!
! Demand[plant,product,month]:
!
!      Jan    Feb    Mar    Apr
!      -----
!
!plant 1:
!      4300,  4200,  6400,  5300,
!      4500,  5400,  6500,  7200,
!      5400,  6700,  7800,  8200,
!
!plant 2:
!      5100,  6200,  5400,  7600,
!      6300,  7100,  5200,  6300,
!      4800,  6500,  5000,  7200,
!
!plant 3:
!      4100,  6100,  4700,  5800,
!      5300,  5200,  5700,  4100,
!      4200,  4100,  5200,  6300,
!
!plant 4:
!      4300,  4100,  5300,  4500,
!      5300,  6400,  4200,  6200,
!      5600,  5200,  3800,  4100
```

When you have three dimensional data in data files you list the data values in same order as the indexes were defined for the data vector in the model. For example, in the above *Demand6* data file the leftmost index is the *plant* index followed by the *product* and the *month* index.

Step 5: Upgrade the 'InvtCost' Data Vector and the 'InvtCapacity' Data Constant to Include the 'plant' Index

Since you can now store inventory at each plant you need to update the inventory cost and the inventory capacity data to include the *plant* index. In the model editor, add the index *plant* to the declaration of the *InvtCost* data vector and the *InvtCapacity* data constant. Then, for the *InvtCost* data vector remove the list of numbers and replace it with the keyword *DATAFILE* and the file name '*InvtCost.dat*'. For the *InvtCapacity* data constant remove the single value 800 and replace it with the list of four values, one for each plant, taken from the problem description earlier in this session.

```
InvtCost[plant, product]      := DATAFILE("InvtCost.dat");
InvtCapacity[plant]          := (800, 400, 500, 400);
```

Next you need to create a new data file called '*InvtCost.dat*' using the cost values from the inventory cost table in the problem description. Enter the data values into the data file as follows:

```
!
! InvtCost.dat - Inventory cost per item a month
!
! InvtCost[plant,product]:
!
!      A1      A2      A3
!      -----
!      8.50,   7.00,   6.50
!      9.80,   9.80,   9.80
!      7.50,   7.50,   7.50
!      9.30,   8.00,   6.50
```

Step 6: Add a Data Vector to Include Shipping Costs

There are certain costs involved in shipping products between plants. In the model editor, add a new data vector called *ShipCost* defined over the two alias indexes *fromplant* and *toplant* followed by the keyword *DATAFILE* and the file name *ShipCost.dat*.

```
ShipCost[fromplant, toplant] := DATAFILE("ShipCost.dat");
```

Next, you need to create a new data file called *ShipCost.dat* containing the cost figures for shipping between the plants provided in the problem description. Enter the data values for the data file as follows:

Part IV A MPL Tutorial

```
!
! ShipCost.dat - Shipping costs from plant to plant
!
! ShipCost[fromplant, toplant]
!
      0, 15.00, 21.00, 13.00,
16.00, 0, 12.00, 12.00,
14.00, 17.00, 0, 15.00,
21.00, 13.00, 10.00, 0
```

Step 7: Add a Variable Vector for Shipments Between Plants

We are allowing shipping between the plants, therefore, you need to create a new variable that decides how much is to be shipped of each product per month. This variable vector will be defined over the alias indexes *fromplant* and *toplant* as well as the *product* and the *month* index. Add the following definition for the *Ship* variable vector to the *VARIABLES* section:

```
Ship[product, month, fromplant, toplant]
  WHERE (fromplant <> toplant);
```

Since we do not want to ship a product from a plant back to the same plant we are using a *WHERE* condition to remove all of the vector elements where the source plant is the same as the destination plant.

Step 8: Add the Total Shipping Cost to the Objective Function

In the *MACROS* section add a new macro definition for the total shipping cost called *TotalShipCost* and update the *TotalCost* macro to include the new macro as follows:

```
TotalShipCost := SUM(product, month, fromplant, toplant: ShipCost * Ship);
TotalCost      := TotalProdCost + TotalInvtCost + TotalShipCost;
```

Note that the actual objective function definition does not need to be changed as the *TotalCost* macro contains all the changes.

Step 9: Upgrade the Inventory Balance Constraint to a Plant Balance Constraint by Adding the 'Ship' Variable

Since we now have shipments allowed between the plants, you need to upgrade the inventory balance constraint from the previous model to a plant balance constraint. First, change the name of the constraint from *InvtBal* to *PlantBal* and add the index *plant* to the declaration. Next, since the constraint is now declared over the index *plant* we do not have to sum over the index *plant* for the *Produce* variable anymore.

```
PlantBal[plant, product, month] -> PBal:
  Produce + Inventory[month-1]
  + SUM(fromplant: Ship[fromplant, toplant:=plant])
=
  Sales + Inventory
  + SUM(toplant: Ship[fromplant:=plant, toplant]);
```

On the left hand side, where we bring together everything that is going into the plant, add a summation to add together all the shipments from each of the other plants to the current plant. Inside the summation, enter the *Ship* variable with the destination plant, taking the value of the current *plant* index.

On the right hand side, where we bring together everything that is going out of the plant add another summation to add together all the shipments from the current plant to each of the other plants. Inside the summation enter the *Ship* variable with the source plant this time taking the value of the current *plant* index for the constraint.

6.5 Solve the Model and Analyze the Solution

The next step is to solve the *Planning6* model by choosing *Solve CPLEX* from the *Run* menu. If everything goes well **MPL** will display the message “*Optimal Solution Found*”. If there is an error message window with a syntax error please check the formulation you entered with the model detailed earlier in this session.

You will use the Model Definitions Window again as in session 5 to look at the parts of the solution that we are interested in. To open the Model Definitions Window for the *Planning6* model choose *Model Definitions* from the *View* menu.

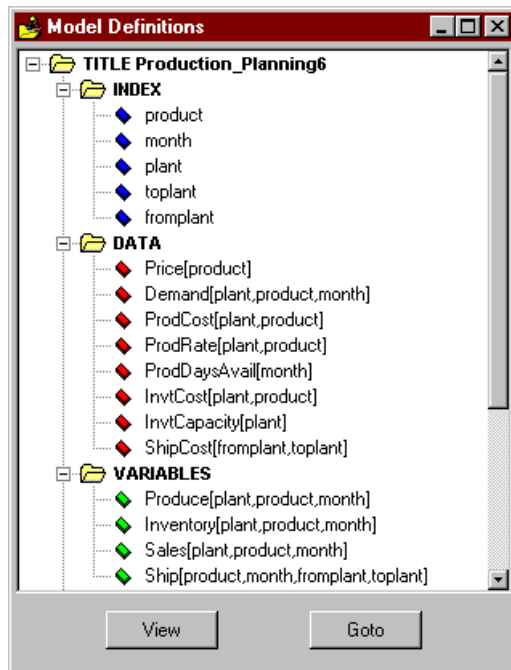


Figure T6.1: The Model Definitions Window for the *Planning6* model

To look at the values for the *Produce* variable, either double click on the *Produce* item in the tree or alternatively select it and then press the *View* button. This will display a window containing only the values for the *Produce* variable shown on the next page.

Session 6 Allow Shipments Between Plants

VARIABLE Produce[plant,product,month] :

plant	product	month	Activity	Reduced Cost
p1	A1	Jan	4300.0	0.0
p1	A1	Feb	4200.0	0.0
p1	A1	Mar	6400.0	0.0
p1	A1	Apr	5300.0	0.0
p1	A2	Jan	1080.0	0.0
p1	A3	Jan	5400.0	0.0
p1	A3	Feb	5220.0	0.0
p1	A3	Mar	4590.0	0.0
p1	A3	Apr	5130.0	0.0
p2	A1	Jan	5100.0	0.0
p2	A1	Feb	6200.0	0.0
p2	A1	Mar	5400.0	0.0
p2	A1	Apr	7600.0	0.0
p2	A2	Jan	6177.3	0.0
p2	A2	Feb	3927.3	0.0
p2	A2	Mar	5931.8	0.0
p2	A2	Apr	3681.8	0.0
p3	A1	Jan	4100.0	0.0
p3	A1	Feb	6100.0	0.0
p3	A1	Mar	4700.0	0.0
p3	A1	Apr	5800.0	0.0
p3	A3	Jan	4166.7	0.0
p3	A3	Feb	1933.3	0.0
p3	A3	Mar	3766.7	0.0
p3	A3	Apr	2733.3	0.0
p4	A1	Jan	3850.0	0.0
p4	A1	Feb	2828.6	0.0
p4	A1	Mar	5300.0	0.0
p4	A1	Apr	4500.0	0.0
p4	A3	Jan	5600.0	0.0
p4	A3	Feb	5200.0	0.0
p4	A3	Mar	4677.3	0.0
p4	A3	Apr	4836.4	0.0

As you can see the production is now distributed between the different plants in a more cost effective manner. Certain products are clearly better to produce at particular plants due to taking into account the production cost and the shipping cost. For example, product A2 is produced in plants p1 and p2, but not p3 and p4, while product A3 is produced in plants p1, p3 and p4. Product A1 is most economically produced in all of the plants.

Part IV A MPL Tutorial

If you go to the tree window again and open up a window for the *Ship* variable you will get the following solution values:

VARIABLE Ship[product,month,fromplant,toplant] :

product	month	fromplant	toplant	Activity	Reduced Cost
A2	Mar	p2	p4	331.8	0.0
A3	Mar	p4	p3	877.3	0.0
A3	Apr	p4	p3	736.4	0.0

As you can see the model proposes that we ship product *A2* from plant *p2* to plant *p4*. In the same manner, product *A3* is shipped from plant *p4* to *p3*. Clearly, plants *p2* and *p4* have extra capacity at lower cost that can be used to produce goods that plants *p4* and *p3* need.

SESSION 7:

Formulating Models With

Sparse Data in MPL

Often, when working with large models, the data for the model tends not to be dense; as in the previous models we worked on, but rather in a sparse format. Dense data can be perceived in the same manner as spreadsheet data. Ordinarily, it is used for data vectors with not more than two dimensions, where every column and every row is filled with data.

Sparse data, on the other hand, typically involves multiple dimensions, but does not necessarily contain values for every combination of the indexes. Sparse data is usually stored in a table format, where each column represents an index or a data value. When working with large sparse data sets it is common to work with the data in a table format as this allows you to easily skip certain combinations of the index, that are not valid, by omitting them in the table.

7.1 New Concepts in this Session

Equipment Indexes

Sometimes, when formulating production planning models, the decision involves which machines to use to produce the products. As all of the machines are not available in every plant, this introduces a sparsity into the model. When we define the data and the variable vectors for the model, we will then utilize that sparsity to ensure that the size of the model does not become too large. This can be accomplished, either by using a standard *WHERE* command on a data vector, or by using the *IN* operator to connect the relevant indexes.

Using the IN Operator

The *IN* operator in **MPL** allows you to select one of the domain indexes from a multi-dimensional index. For example, if you have a multi-dimensional index that specifies which machines are available in which plants, you can use the *IN* operator to sum over all the machines for that particular plant.

```
INDEX
  plant      := (p1, p2, p3, p4);
  machine    := (m11, m12, m13, m21, m22, m31, m32, m41);

  PlantMach[plant,machine] :=
    (p1.m11, p1.m12, p1.m13,
     p2.m21, p2.m22,
     p3.m31, p3.m32,
     p4.m41);
```

In the above example, we have defined a multidimensional index called *PlantMach* that connects the plants to the corresponding machines.

The *PlantMach* index can then be used, selectively, to choose only the machines that are available in a particular plant. For example:

```
SUBJECT TO
  PlantCapacity[plant] :
    SUM(machine IN PlantMach: Produce[machine]) <= MaxCapacity[plant];
```

In the above example, we sum together how much is produced on each machine at that particular plant. Then we make sure that the total production is limited to the maximum capacity.

Index Files

Just as you can store the data in external data files, you can also store indexes in external index files. Index files allow you to store the elements of an index in a file instead of specifying them directly in the model. When you are defining an index with an index file use the keyword *INDEXFILE* with a filename instead of the usual list of elements. For example:

```
INDEX
product := INDEXFILE("Product.idx");
month   := INDEXFILE("Month.idx");
plant   := INDEXFILE("Plant.idx");
```

The index file is just a standard text file containing a list of the index elements for the particular index. You can separate the elements in the file with either a comma, a space, or both. For example here is a sample index file for the *product* index:

```
! Product.idx - Index element for the product index
A1, A2, A3
```

Sparse Data Files

Generally, when working with sparse models, the data involved is quite large and comes from other applications, such as corporate or desktop databases. In previous sessions, the data was typed into the model file or stored in a dense data file. When working with large data sets, you need a more efficient method to import the data into **MPL** from other applications. For this purpose, **MPL** has the ability to read the data from a *sparse data file*. This file allows you to enter the data in a standard table format, which is closer to the actual characteristics of the data, for example, from a relational database. An example of a sparse data file could be as follows:

```
ProdCost[plant, machine, product] := SPARSEFILE("ProdCost.dat");
```

The file *ProdCost.dat* contains the data in column oriented format with the indexes listed in the first three columns and the corresponding data value at the end of each line as follows:

```
p1, m11, A1, 73.30,
p1, m11, A2, 52.90,
p1, m12, A3, 65.40,
.
.
.
p4, m41, A2, 63.30,
p4, m41, A3, 53.80
```

Part IV A MPL Tutorial

Please note, **MPL** allows you also to store multiple data columns in a single sparse data file. You specify which column by adding a comma and the data column number after the file name inside the parentheses.

```
ProdCost[plant, machine, product] := SPARSEFILE("ProdCost.dat", 2);
```

Using sparse data files is common in real-world modeling. These files can end up being quite large, with multiple indexes and containing lots of data. Frequently, you will have multiple index files and sparse data files storing all the data and leaving the model file only to contain the actual model statements, such as the variables, the objective function and the constraints.

7.2 Problem Description: A Planning Model with Multiple Machines at Each Plant

In this session, you will update the model to have multiple machines distributed between the plants. You will use the model you created in session 6, and make the necessary additions and updates to it.

Since we now have different machines within each plant the production cost and the production rate now have different value for each machine. The following is a table with a single line for each plant, machine, product combination that is applicable.

<i>Plant</i>	<i>Machine</i>	<i>Product</i>	<i>ProdCost</i>	<i>ProdRate</i>
<i>p1</i>	<i>m11</i>	<i>A1</i>	\$73.30	500
	<i>m11</i>	<i>A2</i>	\$52.90	450
	<i>m12</i>	<i>A3</i>	\$65.40	550
	<i>m13</i>	<i>A3</i>	\$47.60	350
<i>p2</i>	<i>m21</i>	<i>A1</i>	\$79.00	550
	<i>m21</i>	<i>A3</i>	\$66.80	450
	<i>m22</i>	<i>A2</i>	\$52.00	300
<i>p3</i>	<i>m31</i>	<i>A1</i>	\$75.80	450
	<i>m31</i>	<i>A3</i>	\$50.90	300
	<i>m32</i>	<i>A1</i>	\$79.90	400
	<i>m32</i>	<i>A2</i>	\$52.10	350
<i>p4</i>	<i>m41</i>	<i>A1</i>	\$82.70	550
	<i>m41</i>	<i>A2</i>	\$63.30	400
	<i>m41</i>	<i>A3</i>	\$53.80	350

The production decision, how much we want to produce of each product, needs to take into account that we now have multiple machines. Therefore, you will update the *Produce* variable to include the *machine* index and then use a *WHERE* condition to exclude the elements that are not applicable, such as plant *p1*, machine *m11*, and product *A3*.

7.3 Formulation of the Model in MPL

Listed below is the entire model formulation for *Planning7*. The additions to the model are highlighted in boldface in order to make it easy for you to see the changes from the model in session 6.

```
TITLE
    Production_Planning7;

INDEX
    product      := (A1, A2, A3);
    month        := (Jan, Feb, Mar, Apr);
    plant        := (p1, p2, p3, p4);
    fromplant    := plant;
    toplant      := plant;
    machine      := (m11, m12, m13, m21, m22, m31, m32, m41);

DATA
    Price[product]                := (120.00, 100.00, 115.00);
    Demand[plant, product, month] := DATAFILE("Demand6.dat");
    ProdCost[plant, machine, product] := SPARSEFILE("Produce.dat", 1);
    ProdRate[plant, machine, product] := SPARSEFILE("Produce.dat", 2);
    ProdDaysAvail[month]          := (23, 20, 23, 22);
    InvtCost[product]             := DATAFILE("InvtCost.dat");
    InvtCapacity[plant]           := (800, 400, 500, 400);
    ShipCost[fromplant, toplant]  := DATAFILE ("ShipCost.dat");

VARIABLES
    Produce[plant, machine, product, month] -> Prod
        WHERE (ProdCost > 0);
    Inventory[plant, product, month]          -> Invt;
    Sales[plant, product, month]              -> Sale;
    Ship[product, month, fromplant, toplant]
        WHERE (fromplant <> toplant);

MACROS
    TotalRevenue := SUM(plant, product, month: Price * Sales);
    TotalProdCost := SUM(plant, machine, product, month: ProdCost * Produce);
    TotalInvtCost := SUM(plant, product, month: InvtCost * Inventory);
    TotalShipCost := SUM(product, month, fromplant, toplant: ShipCost * Ship);
    TotalCost     := TotalProdCost + TotalInvtCost + TotalShipCost;

MODEL

    MAX Profit = TotalRevenue - TotalCost;

SUBJECT TO
    ProdCapacity[plant, machine, month] -> PCap:
        SUM(product: Produce / ProdRate) <= ProdDaysAvail;
```


Session 7 Formulating Models with Sparse Data

```
PlantBal[plant, product, month] -> PBal:  
    SUM(machine: Produce) + Inventory[month-1]  
    + SUM(fromplant: Ship[fromplant, toplant:=plant])  
    =  
    Sales + Inventory  
    + SUM(toplant: Ship[fromplant:=plant, toplant]);  
  
MaxInventory[plant, month] -> MaxI:  
    SUM(product: Inventory) <= InvtCapacity;  
  
BOUNDS  
    Sales < Demand;  
  
END
```

7.4 Enter New Elements to the Model Step-by-Step

Step 1: Start MPL and Create a New Model

1. Start the **MPL** application.
2. Choose *File / Open* and open the model from the previous session *Planning6.mpl*.
3. Choose *File / Save As* to save it as a new model file *Planning7.mpl*.

Step 2: Change the Title for the Model

Change the title for the model to reflect that you are working with the *Planning7* model.

```
TITLE
  Production_Planning7;
```

Step 3: Add the 'machine' Index in the Model

In this model, each plant now has multiple machines. To create an index for the machines add the following definition for the *machine* index in the *INDEX* section.

```
INDEX
  product := (A1, A2, A3);
  month   := (Jan, Feb, Mar, Apr);
  plant   := (p1, p2, p3, p4);
  fromplant := plant;
  toplant  := plant;
  machine := (m11, m12, m13, m21, m22, m31, m32, m41);
```

Step 4: Update the 'ProdCost' and the 'ProdRate' Data Vectors to Include the 'machine' Index

The production cost and the production rate now need to include the machine index since we have different data values for each machine. Also, since the data is now sparse, that is not every plant has every machine, you are going to store the data in a *sparse data file*.

Update the definitions for the *ProdCost* and the *ProdRate* data vectors to include the *machine* index and change the filenames to a new sparse data file called *Produce.dat*. For the production cost specify column 1 after the file name and for the production rate specify column number 2.

Session 7 Formulating Models with Sparse Data

```
DATA
Price[product] := (120.00, 100.00, 115.00);
Demand[plant, product, month] := DATAFILE("Demand6.dat");
ProdCost[plant, machine, product] := SPARSEFILE("Produce.dat", 1);
ProdRate[plant, machine, product] := SPARSEFILE("Produce.dat", 2);
ProdDaysAvail[month] := (23, 20, 23, 22);
InvtCost[product] := DATAFILE("InvtCost.dat");
InvtCapacity[plant] := (800, 400, 500, 400);
ShipCost[fromplant, toplant] := DATAFILE("ShipCost.dat");
```

Step 5: Create a Sparse Data File for the Production Cost and Production Rate

Now you need to create the sparse data file *Produce.dat* from the data given in the problem description earlier in this session.

To create the data file for the production cost and production rate open a new editor window for a data file called *Produce.dat* and type in the following:

```
!
! Produce.dat - Production Cost and Rate per item produced
!
! ProdCost[plant, machine, product]:
! ProdRate[plant, machine, product]:
!

    p1,  m11,  A1,  73.30,  500,
    p1,  m11,  A2,  52.90,  450,
    p1,  m12,  A3,  65.40,  550,
    p1,  m13,  A3,  47.60,  350,

    p2,  m21,  A1,  79.00,  550,
    p2,  m21,  A3,  66.80,  450,
    p2,  m22,  A2,  52.00,  300,

    p3,  m31,  A1,  75.80,  450,
    p3,  m31,  A3,  50.90,  300,
    p3,  m32,  A1,  79.90,  400,
    p3,  m32,  A2,  52.10,  350,

    p4,  m41,  A1,  82.70,  550,
    p4,  m41,  A2,  63.30,  400,
    p4,  m41,  A3,  53.80,  350
```

Step 6: Update the Produce Variable Vector to Include the 'machine' Index

The *Produce* variable now needs to have the *machine* index in the declarations as we need to know on which machine each product is produced. Furthermore, since not all of the machines are in every plant we need to exclude the index combinations that are not valid. This is done by using a *where condition* on the *ProdCost* data vector. Only the combinations of indexes are used where *ProdCost* is greater than zero when expanding the *Produce* variable. Enter the changes to the *Produce* variable as follows:

```
VARIABLES
  Produce[plant, machine, product, month] -> Prod
    WHERE (ProdCost > 0);
```

Step 7: Add the 'machine' Index to the Macro for the Total Production Cost

In the macro for the total production cost add the index *machine* to reflect that the *Produce* variable now contains the *machine* index.

```
MACROS
  TotalRevenue := SUM(plant, product, month: Price * Sales);
  TotalProdCost := SUM(plant, machine, product, month: ProdCost * Produce);
  TotalInvtCost := SUM(plant, product, month: InvtCost * Inventory);
  TotalShipCost := SUM(product, month, fromplant, toplant: ShipCost * Ship);
  TotalCost := TotalProdCost + TotalInvtCost + TotalShipCost;
```

Step 8: Update the 'ProdCapacity' Constraint to Include the 'machine' Index

In the declaration for the production capacity constraint the *machine* index must be included since we now have a separate capacity limit for each machine in the plant. Enter the changes to the *ProdCapacity* constraint as follows:

```
SUBJECT TO
  ProdCapacity[plant, machine, month] -> ProdCap:
    SUM(product: Produce/ProdRate) <= ProdDaysAvail;
```

Step 9: Update the Plant Balance Constraint to Sum the Produce Variable Over All the Machines

In the plant balance constraint there is now a separate *Produce* variable for each machine since we need to add together the total production for the particular plant we now need to sum over the machine index when referring to the *Produce* variable. To do this add the following summation to the *PlantBal* constraint:

```
PlantBal[plant, product, month] -> PBal:
    SUM(machine: Produce) + Inventory[month-1]
+ SUM(fromplant: Ship[fromplant, toplant:=plant])
=
    Sales + Inventory
+ SUM(toplant: Ship[fromplant:=plant, toplant]);
```

7.5 Solve the Model and Analyze the Solution

The next step is to solve the *Planning7* model by choosing *Solve CPLEX* from the *Run* menu. If everything goes well **MPL** will display the message “*Optimal Solution Found*”. If there is an error message window with a syntax error please check the formulation you entered with the model detailed earlier in this session.

You will use the Model Definitions Window again, as in session 6, to look at the parts of the solution that we are interested in. To open the Model Definitions Window for the *Planning7* model choose *Model Definitions* from the *View* menu.

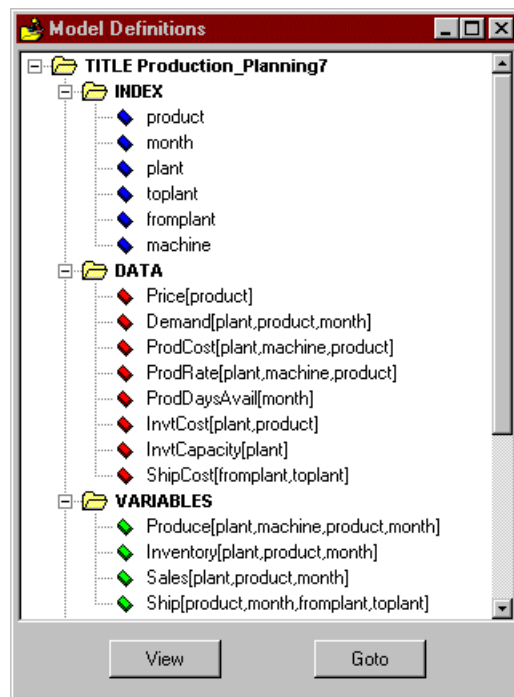


Figure T7.1: The Model Definitions Window for the *Planning7* Model

To look at the values for the *Produce* variable either double click on the produce on the variable the tree or select it and press the *View* button. This will display a view window containing the solution values for only the *Produce* variable which are shown on the next page:

Session 7 Formulating Models with Sparse Data

VARIABLE Produce[plant,machine,product,month] :

plant	machine	product	month	Activity	Reduced Cost
p1	m11	A1	Jan	4300.0000	0.0000
p1	m11	A1	Feb	4200.0000	0.0000
p1	m11	A1	Mar	5487.5000	0.0000
p1	m11	A1	Apr	5300.0000	0.0000
p1	m11	A2	Jan	6480.0000	0.0000
p1	m11	A2	Feb	5220.0000	0.0000
p1	m11	A2	Mar	5411.2500	0.0000
p1	m11	A2	Apr	5130.0000	0.0000
p1	m12	A3	Feb	9049.3506	0.0000
p1	m12	A3	Mar	916.1616	0.0000
p1	m12	A3	Apr	10803.1169	0.0000
p1	m13	A3	Jan	8050.0000	0.0000
p1	m13	A3	Feb	7000.0000	0.0000
p1	m13	A3	Mar	8050.0000	0.0000
p1	m13	A3	Apr	7700.0000	0.0000
p2	m21	A1	Jan	5100.0000	0.0000
p2	m21	A1	Feb	6200.0000	0.0000
p2	m21	A1	Mar	6538.8889	0.0000
p2	m21	A1	Apr	7600.0000	0.0000
p2	m21	A3	Jan	4422.6136	0.0000
p2	m21	A3	Feb	3927.2727	0.0000
p2	m21	A3	Mar	5000.0000	0.0000
p2	m21	A3	Apr	3681.8182	0.0000
p2	m22	A2	Jan	6900.0000	0.0000
p2	m22	A2	Feb	6000.0000	0.0000
p2	m22	A2	Mar	6900.0000	0.0000
p2	m22	A2	Apr	6600.0000	0.0000
p3	m31	A1	Jan	3300.0000	0.0000
p3	m31	A1	Feb	5964.9351	0.0000
p3	m31	A1	Mar	2550.0000	0.0000
p3	m31	A1	Apr	4477.4026	0.0000
p3	m31	A3	Jan	4700.0000	0.0000
p3	m31	A3	Feb	2023.3766	0.0000
p3	m31	A3	Mar	5200.0000	0.0000
p3	m31	A3	Apr	3615.0649	0.0000
p3	m32	A1	Jan	800.0000	0.0000
p3	m32	A1	Feb	135.0649	0.0000
p3	m32	A1	Mar	2150.0000	0.0000
p3	m32	A1	Apr	1322.5974	0.0000
p3	m32	A2	Jan	7350.0000	0.0000
p3	m32	A2	Feb	6881.8182	0.0000
p3	m32	A2	Mar	6168.7500	0.0000
p3	m32	A2	Apr	6542.7273	0.0000
p4	m41	A1	Jan	4300.0000	0.0000
p4	m41	A1	Feb	4100.0000	0.0000
p4	m41	A1	Mar	5073.6111	0.0000
p4	m41	A1	Apr	4500.0000	0.0000
p4	m41	A2	Jan	2270.0000	0.0000
p4	m41	A2	Feb	5018.1818	0.0000
p4	m41	A2	Mar	2500.0000	0.0000
p4	m41	A2	Apr	5527.2727	0.0000
p4	m41	A3	Jan	3327.3864	0.0000
p4	m41	A3	Mar	2633.8384	0.0000

Part IV A MPL Tutorial

The *Produce* variable is now defined over four indexes: *plant*, *machine*, *product* and *month*. For each plant the model decides which machine is the most efficient to produce the products at that particular plant. This table could be used as a basis for a production schedule for the whole company.

The other variable that is interesting in this model is the *Inventory*. If you go to the tree window again and open up a view window for the *Inventory* variable you will get the following solution values:

```
VARIABLE Inventory[plant,product,month] :
```

plant	product	month	Activity	Reduced Cost
p1	A2	Jan	800.0000	0.0000
p2	A2	Jan	400.0000	0.0000
p3	A3	Jan	500.0000	0.0000
p4	A3	Jan	400.0000	0.0000

As you can see the model has decided to produce products *A2* and *A3* during January to ensure we have enough on hand for February.

Most of the plants are now running at full capacity. If you go to the tree window again and open up a view window for the *ProdCapacity* constraint you will get the following solution values:

```
CONSTRAINT ProdCapacity[plant,machine,month] :
```

plant	machine	month	Slack	Shadow Price
p1	m12	Jan	23.0000	0.0000
p1	m12	Feb	3.9595	0.0000
p1	m12	Mar	20.2682	0.0000
p1	m12	Apr	1.2033	0.0000
p2	m21	Jan	3.6947	0.0000

As you can see plant *p1* has some extra capacity for machine *m12* and plant *p2* has some extra capacity for machine *m21*. Other than that all of the machines in every plant is running at full capacity to fulfill demand.

APPENDICES

Appendix A: Character Set

Appendix B: Error Messages

MPL Modeling System



APPENDIX A:

CHARACTER SET

MPL recognizes the following ASCII characters:

Letters

65-90	A..Z	Upper case letters
97-122	a..z	Lower case letters
38	&	Ampersand
39	'	Aphostrophe
64	@	At-sign
95	_	Underscore
96	`	Grave accent

Digits

48-57	0..9	Digits
46	.	Decimal point

White Space and Comments

0-31		Control characters (white space)
32		Space (white space)
33	!	Exclamation point (start a comment)
123	{	Left braces (opens a comment)
125	}	Right braces (closes a comment)

MPL Modeling System

Special Symbols

34	"	Double quotation mark (quoted names)
35	#	Number sign (include commands)
36	\$	Dollar sign (advanced command)
44	,	Comma (item separator)
45,62	->	Becomes (name abbreviation)
46,46	..	Dots (index range)
58	:	Colon (constraint name)
59	;	Semicolon (separator)
63	?	Question mark (interactive data value)
91	[Left bracket (vector index)
93]	Right bracket (vector index)

Arithmetic Operators

37	%	Percent sign
40	(Left parenthesis
41)	Right parenthesis
42	*	Multiplication symbol
43	+	Plus sign
45	-	Minus sign
47	/	Division symbol
94	^	Circumflex

Relational Operators

60	<	Less than
61	=	Equal sign
62	>	Greater than
60,61	<=	Greater than or equal
62,61	>=	Less than or equal

Reserved characters for future use

92	\	Backslash
124		Vertical bar
126	~	Tilde

APPENDIX B:

ERROR MESSAGES

In the presentation of error messages, any text presented as *xxxxx* is a place holder. On your screen, it will be replaced by text or a keyword specific to the error in your formulation.

Format Errors in the Formulation

```
***** A minor mistake was found in line nnn :
```

- 1. The model file must start with a defined keyword, but I found instead '*xxxxx*'.**

The model file must start with one of these keywords:

TITLE, INDEX, DATA, DECISION, VARIABLES, CONSTRAINTS, MACRO,
MODEL, MAX, MIN.

- 2. I expected to see a problem name after the keyword TITLE, but found instead 'xxxx'.**

Example:

```
TITLE + FAME_Simulation ;  
^
```

The problem title can only be a name.

- 3. I expected to see a keyword starting the next section, but found instead 'xxxx'.**

Example:

```
TITLE Plan_for_1_year ;  
  
ProdInvt1 : 20000 + Prod1 = Invt1 + 90000 ;  
^
```

The model file must contain an objective function, which in turn must begin with the keywords *MAX* or *MIN*. In this example the objective function is missing.

- 4. I expected to see a semicolon ';' after the objective function, but found instead 'xxxx'.**

Example:

```
MAX 3x1 + 5x2 { note the lack of semicolon }  
x1 < 4 ;  
^
```

- 5. I expected to see a semicolon ';' after the previous constraint, but found instead 'xxxx'.**

Example:

```

      x1          <  4 ;
      2 x2          < 12 { note the lack of semicolon }
     -3 x1 + 2 x2  < 18 ;
                  ^
    
```

Note that semicolon errors are normally not noticed until the next constraint is parsed. Furthermore, if the next constraint begins with plus or minus sign, the missing semicolon is normally not noticed until the comparison in that constraint has been reached.

- 6. I expected to see either a number or a variable, but found instead 'xxxx'.**

Example:

```

     3x1 + 2x2 + < 18 ;
                ^
    
```

- 7. I expected to see a left parenthesis '(', but found instead 'xxxx'.**

Example:

```

     3x + log 3 < 5 ;
            ^
    
```

A pair of parentheses must be around the argument for the arithmetic functions.

- 8. I expected to see a ')', closing the parenthesis, but found instead 'xxxx'.**

Example:

```

     3(Sb1 + Co1 + So1 = 2 (Sb2 + Co2 + So2) ;
                        ^
    
```

- 9. I expected to see a constraint name after the keyword REFERENCE.**

11. I expected to see a semicolon ';' after the previous bound, but found instead 'xxxx'.

Example:

```
BOUNDS
work1 < 60 {note the lack of semicolon}
work2 < 60 ;
^
```

12. I expected to see a variable name, but found instead 'xxxx'.

Example:

```
BOUNDS
3 < + invt1 ;
^
```

13. I expected to see a relational operator such as '<' or '>', but found instead 'xxxx'.

Example:

```
BOUNDS
x 3 ;
^
```

14. I expected to see a less than operator such as '<', but found instead 'xxxx'.

Example:

```
BOUNDS
3 < x > 4 ;
^
```

15. I expected to see a greater than operator such as '>', but found instead 'xxxx'.

Example:

```
BOUNDS
3 > x = 4 ;
^
```


- 16. A lower bound 'xxxxx' has already been defined for the variable 'xxxxx'.**

Example:

```
BOUNDS
  x > 3;
  4 < x;
```

This is the second time a lower bound is defined for the variable 'x'.

- 17. An upper bound 'xxxxx' has already been defined for the variable 'xxxxx'.**

Example:

```
BOUNDS
  z < 4;
  y < 4;
  z < 3;
```

This is the second time an upper bound is defined for the variable 'z'.

- 18. The lower bound 'xxxxx' is higher than the already defined upper bound 'xxxxx'.**

Example:

```
BOUNDS
  5 < x < 3 ;
      ^
```

- 19. The upper bound 'xxxxx' is lower than the already defined lower bound 'xxxxx'.**

Example:

```
BOUNDS
  x < -5 ;
      ^
```

The upper bound for a variable must be greater than zero unless the variable is given a negative lower bound.

MPL Modeling System

21. **The argument for EXP must be less than 40.0**
22. **The argument for SQR must be less than 10000000000.**
23. **The argument for LN must be greater than zero.**
24. **The argument for LOG must be greater than zero.**
25. **The argument for SQRT must be greater than zero.**
26. **The argument for RANDOM must be a positive integer.**
27. **The random seed must be a positive integer.**
28. **I expected to see an assignment of a new seed value, but found instead 'xxxx'.**
31. **Division by zero in a coefficient is not allowed.**

Example:

```
1/3 prod1 + 2/0 prod2
      ^
```

32. **I expected to see a closing text quote '"' before reaching the end of the line.**

Example:

```
TITLE "Production planning
      ^
```

Text quotes must be closed in the same line.

33. **I found an end of comment '}' without the corresponding beginning '{'.**

- 34. I expected to see a closing braces '}', closing this comment, but reached instead the end of the file.**

Your comments are not paired up properly. Check to make sure that each left braces have an accompanying right braces.

- 35. I expected to see a description of 'xxxxx' after the keyword IS, but found instead 'xxxxx'.**

- 41. The name 'xxxxx' has already been defined.**

- 42. I expected to see an assignment symbol ':=', but found instead 'xxxxx'.**

Example:

```
INDEX
  product  1..3 ;
          ^
```

- 43. I expected to see a name abbreviation, but found instead 'xxxxx'.**

Example:

```
DECISION VARIABLES
  Inventory [month] ->
  Model
  ^
```

- 44. I expected to see an equal sign '=', but found instead 'xxxxx'.**

- 45. I expected to see a colon ':', but found instead 'xxxxx'.**

Example:

```
SUM(i, price*product)
    ^
```

The index list in the summation must be followed with a colon.

- 46. I expected to see a number,
but found instead 'xxxxx'.
- 47. I expected to see a semicolon after the data vector formula,
but found instead 'xxxxx'.
- 48. I expected to see a unit name,
but found instead 'xxxxx'.
- 51. I expected to see a defined index,
but found instead 'xxxxx'.
- 52. I expected to see a defined data vector,
but found instead 'xxxxx'.

Example:

```
INDEX
  i = 1..5;
DECISION VARIABLES
  x[i]
DATA
  A[i] := x ;
          ^
```

- 53. I expected to see a variable vector,
but found instead 'xxxxx'.

Example:

```
INDEX
  i = 1..5;
DATA
  d[i] := (1,2,3,4,5) ;
DECISION VARIABLES ;
  x
BOUNDS
  d[i] < x ;
          ^
```

- 54. I expected to see data or variable vector in the sum, but found instead 'xxxx'.**

Example:

```

DECISION VARIABLES
  x
MAX
  Z = SUM(x)

```

- 55. I expected to see an integer subscript entry for the index 'xxxxx', but found instead 'xxxx'.**

Example:

```

INDEX
  i = 1..5;
MAX
  Z = x[Jan]

```

- 56. I expected to see a an name subscript entry for the index 'xxxxx', but found instead 'xxxx'.**

Example:

```

INDEX
  month := (Jan, Feb, Mar, Apr, May, Jun) ;
MAX
  Z = SUM(month<Dec: Invt) ;

```

- 57. I expected to see an offset value for the index, but found instead 'xxxx'.**

Example:

```

SUM(i, j: x[i-y]);

```

The offset value for a index subscript must be a scalar number, another index or a data vector.

- 58. I expected to see a left bracket '[', but found instead 'xxxx'.**

Example:

```
DATA
  d(i) := (1,2,3,4,5) ;
  ^
```

- 59. I expected to see a right bracket ']', but found instead 'xxxx'.**

Example:

```
DECISION VARIABLES
  production[i -> Prod] ;
  ^
```

- 61. I expected to see a definition such as subscript range for the index 'xxxx', but found instead 'xxxx'.**

Example:

```
INDEX
  i := 1.. ;
  ^
```

- 62. I expected to see a name of a datafile, but found instead 'xxxx'.**

Example:

```
Datafile(DATA):
  ^
```

The data filename must not be keyword such as DATA unless it is put in quotes.

- 63. I expected to see a number element for the data vector, but found instead 'xxxx'.**

- 64. I expected to see a comma separator ',' between data elements, but found instead 'xxxx'.
- 65. There are not enough numbers in this data vector.
- 66. This sparse data element has already been entered.
- 67. I expected to see a field or column number, but found instead 'xxxx'.
- 68. I expected to see a semicolon ';' after the previous macro definition, but found instead 'xxxx'.

Example:

```
MACRO
  A := SUM(i: x)    { Note the lack of semicolon }
  B := SUM(i: y)
  ^
```

- 69. I expected to see the keyword THEN after the IF condition.
- 70. I expected to see the keyword ENDIF at the end of the IF condition.
- 71. The index range is reversed (hi..lo).

Example:

```
INDEX
  product = 3..1 ;
  ^
```

The first number in a index range must be less than or equal to the second one.

- 72. The subscript 'xxxx' is not within the defined range of the index 'xxxx'.

Example:

```
INDEX
  i := 1..5;
  k[i] := (2,4,6);
  ^
```

- 73. The subscript 'xxxx' has already been given for this subindex.**

Example:

```
INDEX
  month := (Jan, Feb, Mar, Apr, May, Jun);
  Holiday[month] := (Apr, May, Jun, Apr);
  ^
```

- 74. This set difference operation failed as this set is not a subset of the earlier.**

- 75. I expected to see an integer sublength, but found instead 'xxxx'.**

Example:

```
INDEX
  month := (Jan, Feb, Mar) : { Note the lack }
DATA
  ^ { of sublength }
```

- 76. This short name list is not the same size as the previous named subscript list (xxxx).**

Example:

```
INDEX
  product := (ProductA, ProductB) -> (A, B, C);
  ^
```

- 77. The index 'xxxx' is not multi-dimensional.**

- 78. Not a valid subscript for the index 'xxxx'.**

- 81. The index 'xxxx' must have the same declaration as the index 'xxxx'.**

- 82. The index 'xxxx' does not match the declaration of the vector 'xxxx'.**

Example:

```
DECISION VARIABLES
  x[j]
MODEL
  MAX Z := SUM(i, j: x[i, j]);
  ^
```


83. The index 'xxxxx' is not specified in the underlying index list.

84. The index 'xxxxx' in vector 'xxxxx' is not specified in the underlying index list.

Example:

```
SUBJECT TO
  constr[i] : x[i,j] < 5:
```

The index 'j' must be specified in the constraint.

85. The fixed subscript 'xxxxx' for the index 'xxxxx' is not possible here as the index has already been referred in the vector.

86. The index 'xxxxx' has already been given a condition where this vector was defined.

87. The option name 'xxxxx' is not recognized by MPL.

91. Could not open the model file *filename* for reading. File not found.

92. The MPL parser run was cancelled by the user.

96. The keyword 'xxxxx' is reserved for use with MPL.

97. The character 'x' is reserved for future use.

These characters are reserved for use in future releases:

\ | ~

98. The keyword 'xxxxx' is reserved for future use.

The following keywords are reserved for use in future releases of **MPL**:

DEFINE, FOR, FOREACH, INIT, INITIAL, INITIALIZE, INF, MOD, NORMAL, POWER, ST, UNIT, UNITS, USING

Errors for Nonlinear Parser

- 111. A nonlinear term was found in a linear (LP) model.**
- 112. A nonlinear term was found in a quadratic objective function.**
- 113. A nonlinear term was found in a linear constraint for QP model.**
- 114. Arithmetic functions is not allowed in a quadratic model.**
- 115. Exponent on a variable is not allowed here.**
- 116. This exponent has too high power for a quadratic model.**

Errors Using Include Files

- 121. The parameter for `#include` should be a filename.
- 122. Include files can not be nested to more than 8 levels.
- 123. Could not open the include file `xxxxx` for reading. File not found.
Most likely, the filename is not a legal filename, or the file can't be found in the current directory.

Errors Using Conditional Directives

- 124. '`xxxxx`' is not a known conditional directive.
The known conditional directives are:
`#define`, `#undef`, `#ifdef`, `#ifndef`, `#ifndef`, `#else`, `#endif`.
- 125. The maximum number of conditional directives is 100.
- 126. This directive has not been defined.
- 127. There is no `#ifdef` for this `#else`.
- 128. There is no `#ifdef` for this `#endif`.

Errors with Data Files

- 141. Cannot open the datafile *filename* for reading. File not found.
- 142. Too few numbers in the datafile *filename*.
- 143. I expected to see a valid integer subscript entry for the index '*xxxx*', but found instead '*xxxx*'.
- 144. I expected to see a valid name subscript entry for the index '*xxxx*', but found instead '*xxxx*'.
- 145. I expected to see the next data value entry, but found instead '*xxxx*'.
- 146. This data element has already been encountered.
- 147. I expected to see a closing text quote (") before reaching the end of the line.

Errors Reading MPS Files

- 151. An MPS file must start with a NAME section.
- 152. An MPS file must have a ROWS section.
- 153. An MPS file must have a COLUMNS section.
- 154. The '*xxxx*' is not a valid row type.
- 155. The '*xxxx*' is not a valid bound.
- 156. The row name '*xxxx*' was not defined in the ROWS section.
- 157. The column name '*xxxx*' was not defined in the COLUMNS section.
- 158. Nonlinear operator '*xxxx*' not recognized.

Errors with Problem Size and Memory

```
***** A problem has come up :
```

- 161. **This problem has too many constraints.
The maximum number is 2100000000.**
- 162. **This problem has too many decision variables.
The maximum number is 2100000000.**
- 163. **This problem has too many nonzeros.
The maximum number is 2100000000.**
- 164. **This problem has too many nonzeros in rim vectors.
The maximum number is 2100000000.**
- 165. **This problem has too many elements in data vectors.
The maximum number is 2100000000.**
- 170. **This problem has too many symbols.
The maximum number is 64000.**
- 171. **This problem has too many named data constants.
The maximum number is 16000.**
- 172. **This problem has too many indexes.
The maximum number is 16000.**
- 173. **This problem has too many data and variable vectors.
The maximum number is 16000.**
- 174. **This problem has too many variable vectors.
The maximum number is 16000.**
- 175. **This problem has too many constraints vectors.
The maximum number is 16000.**

- 176. This problem has too many defined macros.
The maximum number is 16000.**
- 177. This problem has too many special ordered sets.
The maximum number is 16000.**
- 181. This named index has too many subscript elements.
The maximum number is 100000.**
- 182. This subindex has too many subscript elements.
The maximum number is 100000.**
- 183. This index list has too many indexes.
The maximum number is 16.**
- 184. This term has too many index referrals.
The maximum number is 16.**
- 185. The term has too many data vector referrals.
The maximum number is 8.**
- 186. This index formula has too many index referrals.
The maximum number is 4.**
- 191. The multi-dimensional index 'xxxxx' is too large for the 32-bit version of MPL. The maximum size for all the indexes multiplied together is 4294967295 (2^{32}). Please contact Maximal Software for details on how to get a 64-bit version that can handle problems of this size.**
- 192. The vector 'xxxxx' is too large for the 32-bit version of MPL. The maximum size for all the indexes multiplied together is 4294967295 (2^{32}). Please contact Maximal Software for details on how to get a 64-bit version that can handle problems of this size.**
- 199. This machine does not have enough free memory to read in this model.**

Errors Using Database Connection

```
***** A minor mistake was found in line nnn :
```

- 200. This version of MPL does not have the database connection installed.
For more information, please contact Maximal Software.
- 201. I expected to see the index 'xxxxx',
but found instead 'xxxxx'.
- 202. I expected to see either the data vector 'xxxxx' or an index name,
but found instead 'xxxxx'.
- 203. I expected to see either the variable vector 'xxxxx' or an index name,
but found instead 'xxxxx'.
- 204. I expected to see a relational operator,
but found instead 'xxxxx'.
- 205. I expected to see an index from the vector,
but found instead 'xxxxx'.
- 206. I expected to see an import column entry,
but found instead 'xxxxx'.
- 207. I expected to see either INDEX or DATA keyword,
but found instead 'xxxxx'.
- 208. I expected to see the DATABASE keyword,
but found instead 'xxxxx'.
- 211. I expected to see a database table name,
but found instead 'xxxxx'.
- 212. I expected to see a database column name,
but found instead 'xxxxx'.

- 213. The column 'xxxxx' was not found in the database table.
- 214. A column for reading index must be either alpha or numeric.
- 215. A column for reading data must be numeric.
- 216. I expected to see a SQL statement string, but found instead 'xxxxx'.
- 217. Maximum number of exported vectors is 10000.
- 221. Could not initialize 'xxxxx'.
- 222. Paradox error: 'xxxxx'.
- 223. Codebase error: Table 'xxxxx' not found.
- 224. ODBC error: 'xxxxx'
- 225. Oracle error: 'xxxxx'.
- 226. You have not logged on the 'xxxxx' database yet.
- 227. Internal database error: 'xxxxx'.
- 241. Expected to see an Excel Filename or Range specification.
- 242. Expected to see an Excel Range specification.
- 243. Could not start Excel for data import.
- 244. Could not open Excel workbook file 'xxxxx'.
- 245. Could not find sheet 'xxxxx' in Excel workbook 'xxxxx'.
- 246. Could not find range 'xxxxx' in Excel workbook 'xxxxx'.

Index

#

#else, conditional directive.....	151
#endif, conditional directive.....	151
#ifdef, conditional directive	151
#ifndef, conditional directive	151
#undef, conditional directive.....	151

A

Abort if conditions.....	207
using FORSOME keyword in.....	207
About MPL for Windows dialog box	139
ABS function.....	197
Absolute gap	
CPLEX parameter	107
XPRESS parameter.....	125
Access database file, model file option	153
Access, calling MPL from.....	17
Accuracy numbers	196
ACOS function.....	197
ACOSH function	197
Activity keyword	180, 189, 225, 227
Advance basis, CPLEX parameter	91
Advanced starting values, CPLEX parameter.....	101
Algorithm	
CPLEX parameter	87
XPRESS parameter.....	128
Algorithm, CPLEX parameter.....	108
Alias index.....	158
ALL keyword.....	180, 189
AND keyword	161
Application limit, CPLEX parameter	91
ARCTAN function	197
Arithmetic	196
in bounds	191
on data	171, 172
on subscripts	214
Arithmetic functions	197
Arithmetic operators	326
Arrange window icons	135
ASCII characters.....	150, 325
ASIN function	197
ASINH function.....	197
ATAN function.....	197

ATAN2 function.....	197
ATANH function.....	197
Auto save, environment option.....	75

B

Backtrack factor, CPLEX parameter	107
Backup copies, making when installing.....	10
Bakery Model	
Problem Description.....	245
Bakery2.mpl, sample tutorial model.....	248
Bakery2.sol, tutorial solution file.....	252
Balance constraints, MPL Tutorial	270, 271
Barrier cache limit, XPRESS parameter.....	121
Barrier iteration limit, XPRESS parameter.....	121
Barrier memory limit, XPRESS parameter.....	121
Barrier Options for CPLEX dialog box	108
Barrier Options for XPRESS dialog box	128
Barrier progress info log	
CPLEX parameter	94
XPRESS parameter	119
Barrier tolerance, XPRESS parameter.....	123
Basis filename, CPLEX parameter.....	91
Basis interval, CPLEX parameter.....	89
BECOMES keyword.....	166
Becomes operator	
in constraints	187
in variables	178
indexes	157, 165
Best bound interval, CPLEX parameter	99
Binary input file, general solver option.....	86
BINARY keyword	145, 148, 179, 193
Binary output file, general solver option.....	86
Binary variables	145, 179, 193
Block comments, in MPL models	150
Bounds	145, 191
on vector variables.....	191
BOUNDS keyword.....	145, 148, 191
Braces, block comments	150
Branch direction, CPLEX parameter.....	100

C

C/C++, calling MPL from.....	17
Calling MPL from	
C/C++	17

MPL Modeling System

MS Access	17	Conditional indexes	214
Visual Basic	17	Conditions, abort if	207
Candidate pricing list size		Constant value, in the objective function.....	184
CPLEX-MP Parameter	95	Constraint aggregation limit, CPLEX parameter	105
XPRESS parameter	120	Constraint count	
Cascade windows	135	shown in Model Statistics dialog box	65
Case sensitivity		shown in Status Window	27
in the MPL language	77	Constraint equation.....	186, 188
of decision variables.....	198	Constraint vectors, MPL Tutorial.....	257
search and replace.....	49	Constraints.....	145, 185
search for text.....	48	description.....	187
Change filename of solution file	82	formula terms in.....	200
Change options		inventory balance	270, 271
for databases	80	name	185, 187
for generated files.....	84	plain	185
for solution files	82	plant balance	297
for the CPLEX solver.....	87	semicolon.....	185, 186
for the FortMP solver	130	WHERE condition on.....	188
for the MPL environment	75	Constraints count	
for the MPL language.....	77	shown in Model Definitions window	31, 69
for the XA solver.....	130	Contents tab, MPL help system.....	33, 137
for the XPRESS solver	113	Context Sensitive Help, dialog box	34, 242
Change properties for a project.....	55	Control characters, in MPL models.....	150
Change solver options.....	131	Convergence tol, CPLEX parameter	110
Change solver parameter options	130	Convert to minimize, generate MPS option.....	84
Change solver setup.....	133	Copy text to the clipboard	46
Character set.....	149, 325	from the toolbar.....	46
Check sparse data for duplicate entries		Copyright information for MPL	139
data files.....	79	COS function.....	197
MPL Language Option	152	COSH function	197
Check syntax of the model		COUNT keyword.....	203
from the Run menu	57, 249	Cover cuts, CPLEX parameter	103
from the toolbar.....	57	CPLEX parameters	
Cholesky factorisation, XPRESS parameter.....	129	Absolute gap	107
Circular index.....	158	Advance basis	91
Clear current model from memory	60	Advanced starting values	101
Clipboard operations.....	46	Algorithm	87, 108
copy text to the clipboard	46	Application limit	91
cut text to the clipboard	46	Backtrack factor	107
paste text from the clipboard	46	Barrier Options dialog box	108
Clique cuts, CPLEX parameter	103	Barrier progress info log.....	94
Close all windows.....	135	Basis filename.....	91
Close model file.....	39	Basis interval.....	89
Close project file.....	53	Best bound interval.....	99
Coefficient reduction, CPLEX parameter.....	92	Branch direction.....	100
Coefficients	196	Clique cuts	103
in bounds	191	Coefficient reduction.....	92
in formulas	198, 200	Column nonzeros	109
outside parentheses	204	Constraint aggregation limit	105
Column nonzeros, CPLEX parameter	109	Convergence tol.....	110
Command line		Cover cuts	103
GENERATE command	17	Crash method	88
running MPL from	17	Crossover strategy	101
Comments, inserting.....	150	Cutting plane passes limit.....	105
Computational options		Dependency checker.....	92
in General Solver Options dialog box	85	Disjunctive cuts.....	104
Compute ranges.....	83	Feasibility tolerance.....	89
general solver option	86	Flow path cuts	104
solution file option.....	83	Gomory candidate limit	105
Conditional directives	151	Gomory fractional cuts	105

Gomory passes limit.....	105
Growth limit.....	110
Gub cuts.....	104
Implied bound cuts.....	104
Infeasibility finder.....	88
Integrality tolerance.....	107
Iteration limit.....	95
Iteration limit for barrier.....	110
Limit Options dialog box.....	95
Log File Options dialog box.....	93
Lower cutoff.....	106
LP iteration log.....	94
Markowitz tolerance.....	90
Matrix fill limit.....	91
Max correction limit.....	110
Min SOS size.....	96
MIP bound strengthening.....	91
MIP Cuts Options dialog box.....	103
MIP node file.....	97
MIP node log.....	94
MIP priority order.....	100
MIP probe.....	99
MIP solutions limit.....	96
MIP Strategy Options dialog box.....	98
MIP Strategy2 Options dialog box.....	101
MIP strong branching limits.....	97
MIP Tolerance Options dialog box.....	106
MIR cuts.....	104
Network extraction.....	112
Network Options dialog box.....	111
Network pricing.....	111
Node file size limit.....	96
Node limit.....	96
Node selection.....	98
Numeric tolerances.....	112
Object value lower limit.....	96
Object value upper limit.....	96
Objective difference.....	107
Objective range.....	110
Optimality tolerance.....	90
Ordering.....	109
Pass dual problem.....	92
Perturbation.....	89
Perturbation constant.....	89
Perturbation limit.....	89
Preprocessing Options dialog box.....	90
Presolve.....	91
Presolve relaxed LP.....	92
Pricing candidate list size.....	95
Pricing dual.....	89
Pricing primal.....	88
Probing.....	99
Reduced cost fixing.....	101
Refactorization.....	89
Relative gap.....	107
Relative object diff.....	107
Rounding heuristic.....	102
Rounding heuristic frequency.....	102
Row multiplier factor for cuts.....	105
Scaling.....	88
Scan for SOS.....	102
Send iteration log to.....	93
Simplex Options dialog box.....	87
Singularity limit.....	96
Start algorithm.....	102
Starting point algorithm.....	109
Sub algorithm.....	102
Time limit.....	95
Tree memory limit.....	96
Upper cutoff.....	106
Variable selection.....	99
Variable upper.....	110
CPLEX solver, installing.....	236
Crash method	
CPLEX parameter.....	88
XPRESS parameter.....	114
Create a new model file.....	37
Create a new project file.....	52
CREATE keyword.....	225, 227
Cross-over control, XPRESS parameter.....	128
Crossover strategy, CPLEX parameter.....	101
Cut strategy, XPRESS parameter.....	126
Cut text to the clipboard.....	46
from the toolbar.....	46
Cut-off, XPRESS parameter.....	127
Cutting plane passes limit, CPLEX parameter.....	105

D	
Data arithmetic.....	171, 172
Data constants.....	30, 168
entered at run-time by user.....	168
MPL Tutorial.....	257
Data files	
check sparse data for duplicate entries.....	79
export constraint values to.....	189
export variable values to.....	180
input directory.....	79, 152
MPL Tutorial.....	284, 311
output directory.....	79, 152
quicksort for sparse data option.....	79, 153
Data Files, MPL Language Option.....	79
DATA keyword.....	145, 147, 168, 170
Data section.....	145, 167
Data vectors.....	170
import from database.....	176
import from Excel.....	174
MPL Tutorial.....	256
read from sparse data file.....	173
shown in Model Definitions window.....	29, 67
sparse.....	171
Database	
Access database file option.....	153
Directory option.....	153
Excel workbook file option.....	154
ODBC data source option.....	153
Database connection.....	217
default database option.....	80, 153
export constraint values.....	190, 226

MPL Modeling System

export variable values.....	182, 223
import data vectors.....	176, 221
import index values.....	166, 219
messages send to Message Window.....	31, 70, 76
names abbreviation.....	166
where conditions.....	166, 222
DATABASE keyword.....	166, 176, 182, 190, 219
Database Options dialog box.....	80
default database.....	80
directory or database files.....	80
do not export.....	81
ODBC data source.....	80
password.....	81
username.....	81
Database password, model file option.....	154
Database type, model file option.....	153
Database username, model file option.....	154
Data-dependent generation.....	202
Decimals setting for the solution files.....	82
Decision variables	
case sensitivity.....	198
declaring.....	178
section.....	198
DECISION VARIABLES keyword.....	147, 178
Default database.....	80
model file option.....	153
Default model type, MPL Language Option.....	78
Definition part in the MPL model.....	145, 147
Degradation estimate, XPRESS parameter.....	125
Delete text in the model file.....	46
Delimiters.....	150
Dense column removal, XPRESS parameter.....	129
Dense data vectors.....	30
DENSE keyword.....	171
Density of the matrix	
shown in Model Statistics dialog box.....	65
Dependency checker, CPLEX parameter.....	92
Description	
of constraints.....	187
of variables.....	179
Description of menus.....	35
Dialog boxes	
About MPL for Windows.....	139
Barrier Options for CPLEX.....	108
Barrier Options for XPRESS.....	128
Context Sensitive Help.....	34
Data constant entered at run-time.....	168
Database Options.....	80
Environment Options.....	75
Find.....	48
Find and Replace.....	49
FortMP Option Parameters.....	130, 131
General Solver Options.....	85
Generate File Options.....	84
Goto Line.....	50
Insert File.....	42
Limit Options for CPLEX.....	95
Limit Options for XPRESS.....	120
Log File Options for CPLEX.....	93
Log File Options for XPRESS.....	118
MIP Cuts Options for CPLEX.....	103
MIP Cuts Options for XPRESS.....	126
MIP Strategy Options for CPLEX.....	98
MIP Strategy Options for XPRESS.....	124
MIP Strategy2 Options for CPLEX.....	101
MIP Tolerance Options for CPLEX.....	106
Model Statistics.....	65
MPL Error Message.....	57, 249
MPL Help System.....	136
MPL Language Options.....	77
Network Options for CPLEX.....	111
Open File.....	38
Preprocessing Options for CPLEX.....	90
Preprocessing Options for XPRESS.....	116
Print File.....	43
Project New.....	52
Project Open.....	52
Project Properties.....	55
Project Save As.....	54
Save As File.....	40
Save Question.....	39, 44
Save Selection.....	41
Simplex Options for CPLEX.....	87
Simplex Options for XPRESS.....	113
Solution File Options.....	82
Solver Menu Setup.....	132
Solver Options List.....	131
Solver Setup Options.....	133
Status Window.....	58
Tolerance Options for XPRESS.....	122
View Other Files.....	63
XA Option Parameters.....	130
Direct assignment of subscripts.....	215
Directories	
database files.....	80, 153
input data files.....	152
model file option.....	153
opening file.....	38, 63
opening project file.....	53
output data files.....	152
save file.....	40, 54
Disjunctive cuts, CPLEX parameter.....	104
Distribution models, MPL Tutorial.....	296
DLL solvers.....	58
Domain indexes.....	162, 210
Domains	
defining for the model.....	155
DOS legacy solvers.....	58, 132
Dot operator	
set domain index.....	212
Double quotes, in names.....	149
Double-click as goto in model	
for model definitions window.....	76
set in environment options dialog box.....	76

E

Edit menu.....	45
copy text to the clipboard.....	46

- cut text to the clipboard 46
- delete text in the model file 46
- paste text from the clipboard 46
- undo changes in the model 45
- Editing model file 45
- Editor
- clipboard operations 46
 - close model file 39
 - copy text to the clipboard 46
 - create a new model file 37
 - cut text to the clipboard 46
 - delete text in the model file 46
 - goto line in the model 50
 - insert file into the editor 42
 - open model file 38
 - paste text from the clipboard 46
 - print file 43
 - replace text in the model 49
 - save model file 40
 - save selected text to a file 41
 - search for text in the model 48
 - undo changes 45
- Element count
- shown in Model Statistics dialog box 65
- ELSE keyword 202
- Empty names 149
- END keyword 145, 148
- Environment options
- for the Message Window 31, 70
 - for the Model Definitions window 28, 66
- Environment Options dialog box 75
- auto save model file 75
 - database connection to Message Window 76
 - double-click as goto in model 76
 - error messages to Message Window 76
 - expand branches 76
 - MPL input lines to Message Window 76
 - performance statistics to Message Window 76
 - short filenames in window titles 75
 - show element count 76
 - solver iteration log to Message Window 76
 - SQL statements to Message Window 76
 - status messages to Message Window 76
 - warning messages to Message Window 76
- Environment, using the MPL 24
- Equal constraints 186
- Equipment indexes, MPL Tutorial 310
- Error Message Window 27
- checking syntax of the model 57, 249
- Error messages 27, 327
- errors reading MPS files 342
 - errors using conditional directives 341
 - errors using include files 341
 - errors using nonlinear parser 340
 - errors with data files 342
 - errors with database connection 345
 - errors with problem size 343
 - format errors in the formulation 327
 - send to Message Window 31, 70, 76
- Estimate deg. mult, XPRESS parameter 125
- Excel
- calling MPL from 17
 - export constraint values 189
 - export variable values 181
 - import data vectors 174
 - import index values 165
 - skip over empty, model file option 154
 - workbook file, model file option 154
 - worksheet name, model file option 154
- Excel workbook file, model file option 154
- EXCELLIST keyword 174, 181, 189
- EXCEL RANGE keyword 165, 174, 181, 189
- EXCELPARSE keyword 174, 181, 189
- EXCEPT keyword, in WHERE conditions 203
- Exclamation mark, single line comment 150
- Exit MPL 44
- EXP function 197
- Expand branches, in model definitions window
- environment option 76
- Expansion
- of variables 179
 - of vectors 200
- Export to data file
- constraint values 189
 - variable values 180
- Export to database
- constraint values 190, 226
 - create option 225, 227
 - do not export option 81
 - refill option 225, 227
 - variable values 182, 223
- Export to Excel
- constraint values 189
 - variable values 181
- EXPORT TO keyword 180, 189, 223, 226
- External data files
- MPL Tutorial 284
- External index files 164
- Extra processors for parallel MIP, XPRESS parameter 121
-
- F**
- Feasibility tolerance, CPLEX parameter 89
- File margins
- for generate file 84
- File menu 37
- close model file 39
 - create a new model file 37
 - exit MPL 44
 - insert file into the editor 42
 - open model file 38
 - print model file 43
 - save model file 40
 - save selected text to a file 41
- File Open dialog box 38
- from the toolbar 38
- Filename, in the solution file 82
- Files

close model file	39
close project file	53
create a new model file	37
create a new project file	52
include files	151
insert file into the editor	42
MPS generate options	84
open model file	38
open project file	52
opening view file	63
save model file	40
save project file	54
save selected text to a file	41
shown in a View window	62
Find and Replace dialog box	49
from the toolbar	49
Find and replace text in the model	49
Find dialog box	48
from the toolbar	48
Find tab, MPL help system	33, 138
Find text in the model	48
FIRST keyword	203
Fixed subscripts	199
Fixed variables	191
Flow path cuts, CPLEX parameter	104
Folders	
opening file	38, 63
opening project file	53
save file	40, 54
Formulas	196
building	200
IF/IFF conditions on formula terms	202
in constraints	186
indexes referred in	203
using parentheses in	204
where conditions on formula terms	202
Formulating the model	178
identify constraints	247
identify decision variables	246
identify objective function	246
MPL Tutorial	243
FORSOME keyword	207
in Abort if conditions	207
FortMP Option Parameters dialog box	130
FortMP solver	
change options	130
FoxPro, calling MPL from	17
Fractions	196
FREE keyword	145, 148, 192
Free variables	145, 192
Function indexes	160
Functions, arithmetic	197

G

General solver option, infeasibility finder	86
General Solver Options dialog box	85
compute ranges	86
generate binary input file	86

generate binary output file	86
generate files	86
generate MPS file	86
generate native input file	86
generate output file	86
log filename	86
send iteration log to	
log file	86
message window	86
solution mapping file	86
use advance basis	85
use infeasibility finder	86
Generate File Options dialog box	84
convert to minimize	84
generate file margins	84
generate MPS files options	84
include comments	84
integer variables in MPS files	85
single column	84
SOS reference rows	84
Generate files	
from the command line	17
general solver options	86
input file	60
margins	84
solution file	59, 82
Generation of formula terms	
where conditions	202
Gomory candidate limit, CPLEX parameter	105
Gomory cuts, XPRESS parameter	127
Gomory fractional cuts, CPLEX parameter	105
Gomory passes limit, CPLEX parameter	105
Goto Line dialog box	50
from the toolbar	50
Goto line in the model	50
Graph menu	71
graph of the matrix	71
graph of the objective function	73
Graph of the matrix	71
Graph of the objective function	73
Graph windows, zooming	72
Greater than constraints	186
Growth limit, CPLEX parameter	110
Gub cuts, CPLEX parameter	104

H

Hardware, system requirements	10
Help menu	136
about MPL for Windows	139
help topics	136, 241
search for help on	136
Help system	24
contents tab	33, 137
context sensitive	34, 242
covered in MPL Tutorial	241
find tab	33, 138
for MPL	136
index tab	33, 138

I

Identify model elements	
constraints	247
decision variables	246
objective function	246
IF keyword	202
IF/IIF conditions	
on formula terms	202
IIF keyword	202
Immediate If function (IIF)	202
Implied bound cuts, CPLEX parameter	104
Import from database	
Access database file option	153
data vectors	176, 221
default database option	80, 153
directory option	80, 153
Excel workbook file option	154
indexes	164, 219
ODBC data source option	80, 153
password option	81, 154
username option	81, 154
Import from Excel spreadsheet	174
Import from spreadsheet	
Excel skip over empty option	154
Excel workbook file option	154
Excel worksheet name option	154
IN operator, set membership	210
Include comments, generate MPS option	84
Include directives	151
Include files	151
INDEX keyword	145, 147, 156
Index operations	
defining sets using	161
difference	161
intersection	161
union	161
use of NOT	161
Index section	155
Index tab, MPL help system	33, 138
Indexed bounds	192
Indexed names, name generation	79, 153
Indexed variable	199
Indexes	
alias	158
circular	158
conditional	214
connecting together	162
defined as list of names	157
defined as list of numbers	159
domain	210
domain index of	162
fixed subscript	199
functions in MPL	160
import from database	166
import from Excel	165
MPL Tutorial	256
multi-dimensional	162
parent index of	162
read from external files	164

referred in formulas	203
shown in Model Definitions window	29, 67
using subsets in MPL	159, 161
Indexes and Vectors	
MPL Tutorial	255
INDEXFILE keyword	164
Infeasibility Sets, XPRESS parameter	114
Infeasibility finder	
general solver option	86
Infeasibility finder, CPLEX parameter	88
INITIAL keyword	179
Initial solution, XPRESS parameter	114
Initial values for variables	179
Input directory for data files	79, 152
Input file	
shown in a View window	62
Input filenames, for DOS solvers	134
Insert File dialog box	42
Insert file into the editor	42
Installing MPL	
making backup copies	10
Mplwin directory	10
no solvers available	236
setup solvers	236
Windows directory	10
Windows system directory	10
Integer functions	
using in formula terms	203
Integer indexes	155
within a range	156
INTEGER keyword	145, 148, 179, 193
Integer markers, MPS files	85
Integer variables	145, 179, 193
define in MPS file	85
Integrality tolerance, CPLEX parameter	107
Integrated Environment	
Main window	19
Integrated environment options	75
Interactive data	168
INTERSECTION keyword	161
Inventory balance constraints, MPL Tutorial	270, 271
Inventory variables, MPL Tutorial	270
IS keyword	179
Iteration limit for barrier, CPLEX parameter	110
Iteration limit, CPLEX parameter	95

K

Keywords	
Activity	180, 189, 225, 227
ALL	180, 189
AND	161
BECOMES	166
BINARY	148, 179, 193
BOUNDS	148, 191
COUNT	203
CREATE	225, 227
DATA	147, 168, 170
DATABASE	166, 176, 182, 190, 219

DECISION VARIABLES	147, 178
DENSE	171
ELSE	202
END	145, 148
EXCELLIST	174, 181, 189
EXCEL RANGE	165, 174, 181, 189
EXCELSPARSE	174, 181, 189
EXCEPT	203
EXPORT TO	180, 189, 223, 226
FIRST	203
FORSOME	207
FREE	148, 192
IF	202
IIF	202
INDEX	147, 156
INDEXFILE	164
INITIAL	179
INTEGER	148, 179, 193
INTERSECTION	161
LAST	203
MACROS	147, 183, 206
MAX	148, 184
MAXIMIZE	184
MIN	148, 184
MINIMIZE	184
MODEL	145, 148
NAMED	157
NOT	161
ObjectCoeff	180, 225
ObjectLower	180, 225
ObjectUpper	180, 225
OR	161
OVER	213
ReducedCost	180, 225
REFILL	225, 227
RhsLower	189, 227
RhsUpper	189, 227
RhsValue	189, 227
SEMICONT	192
ShadowPrice	189, 226, 227
Slack	189, 227
SOLUTIONFILE	180, 189
SOS	148, 193
SPARSE	171
SPARSEFILE	173, 180, 189
SUBJECT TO	145, 148, 185
SUM	205
THEN	202
TITLE	147
UNION	161
VARIABLES	178
WHERE	166, 180, 202, 222

L

LAST keyword	203
Less than constraint	186
Level and frequency, XPRESS parameter	127
Licensing information for MPL	139

Lifted cover inequalities, XPRESS parameter	127
Limit Options for CPLEX dialog box	95
Limit Options for XPRESS dialog box	120
Linear	
model type	153
Linear models, model type	78
List of open windows	135
Load model file into editor	38
MPL Tutorial	235
Location indexes, MPL Tutorial	284
Log file	
CPLEX parameter	93
send iteration log to	86
XPRESS parameter	118
Log File Options for CPLEX dialog box	93
Log File Options for XPRESS dialog box	118
Log filename, general solver option	86
LOG function	197
LOG10 function	197
Logical preprocessing, XPRESS parameter	117
Lower bounds defining in MPL	191
Lower cutoff, CPLEX parameter	106
LP Iteration Limit, XPRESS parameter	120
LP iteration log	
CPLEX parameter	94
XPRESS parameter	119

M

Macro section	145, 183
Macros	
name	183
referring to macros in formulas	206
semicolon	183
shown in Model Definitions window	30, 69
MACROS keyword	145, 147, 183, 206
Main menu	36
Main Window in MPL	19
Markowitz tolerance	
CPLEX parameter	90
XPRESS parameter	123
Matrix fill limit, CPLEX parameter	91
Matrix Graph	71
coloring of	72
spreadsheet view	72
zooming	72
matrix multiplication	215
Max correction limit, CPLEX parameter	110
Max cuts, XPRESS parameter	127
MAX keyword	145, 148, 184
Max nonzero coeffs, XPRESS parameter	127
Max subscript length	157, 165, 166
name generation	79, 153
Max variable length, name generation	79, 153
MAXIMIZE keyword	184
Maximum problem size for MPL	139
Memory usage	144
shown in Status Window	26
Menus	

Edit	45
File	37
Graph	71
Help	136
Main	19
Options	74
Project.....	51
Run	56
Search	47
Solve	133
View	61
Window	135
Message line, in the status window	26
Message window	70
change options for	31, 70, 76
database connection.....	31, 70
error messages.....	31, 70
MPL input lines.....	31, 70
Options dialog box	76
performance statistics.....	31, 70
send iteration log to	86
solver iteration log.....	31, 70
SQL statements	31, 70
Status Window messages	31, 70
warning messages	31, 70
Message window, CPLEX parameter.....	93
MIN keyword	145, 148, 184
Min SOS size, CPLEX parameter	96
MINIMIZE keyword.....	184
Minor mistakes in the formulation	327
MIP bound strengthening, CPLEX parameter	91
MIP Cuts Options for CPLEX dialog box	103
MIP Cuts Options for XPRESS dialog box	126
MIP node file, CPLEX parameter	97
MIP node log	
CPLEX parameter	94
XPRESS parameter.....	119
MIP priority order, CPLEX parameter	100
MIP probe, CPLEX parameter	99
MIP solutions limit	
CPLEX parameter.....	96
XPRESS parameter.....	121
MIP Strategy Options for CPLEX dialog box.....	98
MIP Strategy Options for XPRESS dialog box	124
MIP Strategy2 Options for CPLEX dialog box.....	101
MIP strong branching limits, CPLEX parameter	97
MIP Tolerance Options for CPLEX dialog box	106
MIR cuts, CPLEX parameter	104
Mistake in model, correcting.....	27
Model	
check syntax of the model	57, 249
solve the model	58, 237
Model definitions window	28, 66
change options for	76
constraints branch described.....	31, 69
data branch described.....	29, 67
index branch described.....	29, 67
macros branch described	30, 69
MPL Tutorial	239
set environment options.....	28, 66
variables branch described.....	30, 68
Model editor	
clipboard operations	46
close model file	39
copy text to the clipboard	46
create a new model file.....	37
cut text to the clipboard	46
delete text in the model file	46
goto line in the model.....	50
insert file into the editor	42
open model file.....	38, 235
paste text from the clipboard	46
print file	43
replace text in the model.....	49
save model file	40
save selected text to a file	41
search for text in the model	48
undo changes.....	45
Model file option settings	
Access database file	153
check sparse data for duplicate entries	152
data files input directory	152
data files output directory	152
default database.....	153
directory for database files.....	153
Excel skip over empty option	154
Excel workbook file.....	154
Excel worksheet name.....	154
max subscript length	153
max variable length	153
name generation options.....	153
name model type	153
ODBC data source	153
password	154
plain variables must be defined	152
username.....	154
Model formulation	
Tutorial Session 2, Bakery2.mpl.....	248
Tutorial Session 3, Planning3.mpl	259
Tutorial Session 4, Planning4.mpl	273
Tutorial Session 5, Planning5.mpl	286
Tutorial Session 6, Planning6.mpl	300
Tutorial Session 7, Planning7.mpl	314
Model info	
in solution file	83
MODEL keyword	145, 148
Model part	
in the MPL model.....	145
overview.....	148
Model Statistics dialog box	65
constraint count.....	65
density of the matrix.....	65
element count.....	65
nonzero count of the matrix	65
objective coefficients.....	65
parsing time	65
RHS count.....	65
rim element count.....	65
variable count.....	65
Model type	

MPL Modeling System

linear	153
linear models	78
nonlinear	152, 153
nonlinear models	78
quadratic models	78
quadratic	153
Model1.mpl, tutorial model session 1	235
Modeling environment	19
Models with sparse data	
MPL Tutorial	309
MPL	
About MPL dialog box	139
copyright information	139
environment	24
help system	136, 241
licensing information	139
Main menu	19
maximum problem size	139
release number	139
serial number	139
MPL environment, using the	12
MPL Error Message Dialog Box	57, 249
MPL input lines	
environment options dialog box	76
send to Message Window	31, 70, 76
MPL Language Options dialog box	77
case sensitive	77
data files	79
Default Model Type	78
log to file	78
log to message window	78
max subscript length	79
name generation options	79
plain variables must be defined	77
MPL quitting, Save Question dialog box	44
MPL running	
from shortcut	16
from start menu	16, 234
MPL Tutorial	
Session 1	233
Session 2	243
Session 3	255
Session 4	269
Session 5	283
Session 6	295
Session 7	309
Mplwin directory	
installing to	10
MPS file generated, general solver option	86
MPS files	
generate options	84
integer markers	85
integer variables	85
options for generate	84
UI bound entries	85
MPS names	
use in the solution file	82
MS Access, calling MPL from	17
Multi-dimensional indexes	29, 67, 162
WHERE condition on	162

Multiple plants, MPL Tutorial	283
Multiple time periods, MPL Tutorial	269
Multiplication of formula terms	200
multiplication of matrixes	215

N

Name generation	
indexed names	79, 153
max subscript length	79, 153
max variable length	79, 153
numeric names	79, 153
prefixed numeric names	79, 153
Name of the problem	147
Named indexes	29, 67, 155, 157
NAMED keyword	157
Names	149
abbreviating	157, 165, 166
empty	149
of constraints	185
of macros	183
of objective function	184
of variables	178
of vector constraints	187
using underscore	149
Native input file generated, general solver option	86
Network extraction, CPLEX parameter	112
Network Options for CPLEX dialog box	111
Network pricing, CPLEX parameter	111
New Concepts	
alias indexes	296
balance constraints	297
data constants	257
data files	311
data, variable, constraint vectors	256
equipment indexes	310
external data files	284
IN operator	310
index files	311
indexes as domains	256
inventory balance constraints	270, 271
periods indexes	270
plant balance constraints	297
plant index	284
sales and inventory variables	270
transportation models	296
transshipment models	296
using summations over vectors	257
where conditions	296
New model file, creating	37
Node file size limit, CPLEX parameter	96
Node limit	
CPLEX parameter	96
XPRESS parameter	121
Node selection	
CPLEX parameter	98
XPRESS parameter	125
Nodeset selection, XPRESS parameter	124
Nonlinear	

initial values	179
model type.....	152, 153
Nonlinear models, model type.....	78
Nonlinear solvers	26
Nonzero count of the matrix	
shown in Model Statistics dialog box	65
Nonzero values only, in the solution file	82
NOT keyword.....	161
Number	149, 196
Number width, setting for solution file.....	82
Numeric indexes.....	29, 67, 155
Numeric names, name generation	79, 153
Numeric tolerances, CPLEX parameter	112
Numerical tolerance, XPRESS parameter	122

O

Object value lower limit, CPLEX parameter	96
Object value upper limit, CPLEX parameter	96
ObjectCoeff keyword.....	180, 225
Objective difference, CPLEX parameter.....	107
Objective function	145
formulas terms in	200
graph of the	73
name of the	184
using constant values in.....	184
Objective function coefficients	
in the solution file.....	83
shown in the Model Statistics dialog box.....	65
Objective range, CPLEX parameter	110
Objective ranges	
in solution file	83
shown in a View window	64
ObjectLower keyword.....	180, 225
ObjectUpper keyword.....	180, 225
ODBC data source	
database option.....	80
model file option	153
Offset value for subscripts	199
On-line help.....	24
Open an existing project file	52
Open File dialog box.....	38
from the toolbar.....	38
Open model file	38
MPL Tutorial	235
Open Project dialog box.....	52
from the toolbar.....	52
Optimality tolerance, CPLEX parameter.....	90
Option settings in MPL.....	152
Options dialog box	
Message window.....	76
Options menu	74
CPLEX barrier options	108
CPLEX limit options	95
CPLEX log file options	93
CPLEX MIP cuts options	103
CPLEX MIP strategy options	98
CPLEX MIP Strategy2 options.....	101
CPLEX MIP tolerance options	106

CPLEX network options.....	111
CPLEX preprocessing options.....	90
CPLEX simplex options	87
database options	80
environment options	75
generate file options	84
MPL language options.....	77
setup options for solvers.....	133
setup solvers for the Run menu	132
solution file options	82
solver menu setup.....	58
solver options list	131
solver parameter options.....	130
XPRESS barrier options.....	128
XPRESS limit options.....	120
XPRESS log file options	118
XPRESS MIP cuts options	126
XPRESS MIP strategy options	124
XPRESS preprocessing options.....	116
XPRESS simplex options	113
XPRESS Tolerance options	122
Options Menu	
general solver options	85
OR keyword.....	161
Ordering algorithm, XPRESS parameter.....	129
Ordering, CPLEX parameter.....	109
Output directory for data files	79, 152
Output file generated, general solver option	86
Output file, shown in a View window	62
Output filenames, for DOS solvers	134
OVER keyword	213
OVER operator, defining set subsets.....	213

P

Paradox, calling MPL from.....	17
Parent index of multi-dimensional index.....	162
Parentheses	204
nested.....	204
Parse model into memory	59
Parsing time	
shown in Model Statistics dialog box	65
Part. pricing cand. list sizing, XPRESS parameter .	120
Pass dual problem, CPLEX parameter	92
Password, model file option	154
Paste text from the clipboard.....	46
from the toolbar.....	46
Pause after solve for DOS solvers	134
Percentage, entering in MPL model	196
Performance statistics	
send to Message Window	31, 70, 76
Period indexes, MPL Tutorial	270
Perturbation	
CPLEX parameter	89
XPRESS parameter	114
Perturbation constant, CPLEX parameter	89
Perturbation limit, CPLEX parameter	89
PI function	197
PIF filenames, for DOS solvers.....	134

MPL Modeling System

Plain constraints, defining in MPL.....	185
Plain variables must be defined	
MPL language option	77, 152
Planning3.mpl, tutorial model file.....	259
Planning3.sol, tutorial solution file.....	265
Planning4.mpl, tutorial model file.....	273
Planning4.sol, tutorial solution file.....	279
Planning5.mpl, tutorial model file.....	286
Planning6.mpl, tutorial model file.....	300
Planning7.mpl, tutorial model file.....	314
POWER function.....	197
Prefixed numeric names, name generation	79, 153
Preprocessing Options for CPLEX dialog box	90
Preprocessing Options for XPRESS dialog box	116
Presolve	
CPLEX parameter	91
XPRESS parameter.....	116
Presolve relaxed LP, CPLEX parameter.....	92
Pricing candidate list size, CPLEX parameter	95
Pricing dual, CPLEX parameter.....	89
Pricing primal, CPLEX parameter	88
Pricing, XPRESS parameter	114
Print File dialog box.....	43
from toolbar	43
Print model file	43
Probing	
CPLEX parameter	99
XPRESS parameter.....	117
Problem Description	
Multi-Period Production Planning	272
Planning Model with Multiple Machines	313
Prod. Planning with Multiple Plants	285
Product-Mix Model.....	258
Shipments Between Plants	298
Tutorial Session 2.....	245
Problem title	147
Product-mix model, MPL Tutorial	245
Project files	
change properties	55
close project file	53
create a new project file.....	52
open project file	52
save project file.....	54
save under different name	54
Project menu.....	51
change properties for a project	55
close project file	53
create a new project file.....	52
open project file	52
save project file.....	54
Project New dialog box.....	52
Project Open dialog box.....	52
from the toolbar.....	52
Project Properties dialog box	55
Project Save	
from the toolbar.....	54
under different name	54
Project Save As dialog box	54
Projects to manage models.....	32
Properties for a project	
dialog box described.....	55
Pseudo cost, XPRESS parameter	125
Pull-down menus in MPL, overview	20
full description of each menu item.....	35
<hr/>	
Q	
Quadratic models, model type.....	78
Quadratic	
model type.....	153
Quicksort data files, for sparse data option.....	153
Quicksort for sparse data, data files	79
Quit MPL	44
<hr/>	
R	
RANDOM function	197
Range, for numeric index.....	156
Ranges	
compute	83
in the solution file.....	83
Ranges objective	
in solution file	83
shown in a View window	64
Ranges RHS	
in solution file	83
shown in a View window	64
Reduced cost	
in solution file	83
shown in a View window	64
Reduced cost fixing	
CPLEX parameter	101
XPRESS parameter.....	117
ReducedCost keyword	180, 225
Refactorization, CPLEX parameter.....	89
REFILL keyword	225, 227
Relational operators	326
Relative gap	
CPLEX parameter	107
XPRESS parameter	125
Relative object diff, CPLEX parameter.....	107
Relaxed LP presolve, CPLEX parameter.....	92
Release number for MPL	139
Replace text in the model.....	49
Reserved characters	326
RHS count	
shown in Model Statistics dialog box	65
RHS ranges	
in solution file	83
shown in a View window	64
RHS values	
in solution file	83
RhsLower keyword	189, 227
RhsUpper keyword	189, 227
RhsValue keyword.....	189, 227
Rim element count	
shown in Model Statistics dialog box	65
Rounding heuristic frequency, CPLEX parameter..	102

- Rounding heuristic, CPLEX parameter 102
- Row multiplier factor for cuts, CPLEX parameter.. 105
- Run Check Syntax 57
- from the toolbar..... 57
- MPL Tutorial 249
- Run menu 56
- check syntax of the model 57
- clear current model from memory 60
- generate input file..... 60
- generate solution file 59
- parse model into memory 59
- setup solvers for 58
- solve current model 59
- solve the model 58
- solver setup 133
- Run Solve 58
- from the toolbar..... 58
- Running MPL
- from shortcut..... 16
- from Start menu 16, 234
- MPL Tutorial 233
- overview 24
-
- S**
- Sales variables, MPL Tutorial 270
- Sample tutorial model
- Session 1, Model1.mpl 235
- Session 2, Bakery2.mpl..... 248
- Session 3, Planning3.mpl 259
- Session 4, Planning4.mpl 273
- Session 5, Planning5.mpl 286
- Session 6, Planning6.mpl 300
- Session 7, Planning7.mpl 314
- Save As File dialog box 40
- from the toolbar..... 40
- Save As Project dialog box 54
- Save model file 40
- from the toolbar..... 40
- Save project file
- from the toolbar..... 54
- under different name 54
- Save Question dialog box
- close model file 39
- quitting MPL..... 44
- Save selected text to a file 41
- Save Selection dialog box 41
- Scalability of MPL..... 144
- Scalar values..... 168
- Scaling
- CPLEX parameter 88
- XPRESS parameter 115
- Scan for SOS, CPLEX parameter 102
- Scientific notation 149
- Search and Replace dialog box 49
- from the toolbar..... 49
- Search and replace text in the model 49
- Search Find dialog box 48
- from the toolbar..... 48
- Search for text in the model 48
- Search menu 47
- find text in the model 48
- goto line in the model 50
- replace text in the model..... 49
- Semicolons
- in bounds..... 191
- in constraints 185, 186
- in macros..... 183
- in objective function..... 184
- in title 147
- in variables 178
- SEMICONT keyword 192
- Semi-continuous variables 192
- Send iteration log to
- CPLEX parameter 93
- general solver option 86
- XPRESS parameter 118
- Sensitivity analysis, general solver option 86
- Serial number for MPL 139
- Session 1, MPL Tutorial
- Running MPL on a Sample Model 233
- Session 2, MPL Tutorial
- Formulating a Simple Product-Mix Model 243
- Session 3, MPL Tutorial
- Introducing Vectors and Indexes 255
- Session 4, MPL Tutorial
- Multiple Time Periods..... 269
- Session 5, MPL Tutorial
- Multiple Plants 283
- Session 6, MPL Tutorial
- Shipments Between Plants 295
- Session 7, MPL Tutorial
- Models with Sparse Data..... 309
- Set domain index with the Dot operator 212
- Set membership with the IN operator..... 210
- Set operations
- defining sets using..... 161
- difference 161
- intersection..... 161
- not 161
- union 161
- Set subsets with the OVER operator 213
- Sets, in MPL models 162
- Setup solvers 133
- Shadow prices
- in solution file 83
- shown in a View window 64
- ShadowPrice keyword..... 189, 226, 227
- Shipments between plants
- MPL Tutorial..... 295
- Short filenames in window titles
- environment option 75
- Shortcut for MPL..... 16
- Show element count, in model definitions window
- environment option 76
- SIGN function 197
- Simple constraints..... 185
- Simple term 200
- Simplex Options for CPLEX dialog box 87

MPL Modeling System

Simplex Options for XPRESS dialog box	113	SOS variables	193
SIN function	197	Space character, in MPL models	150
Single column, generate MPS option	84	Sparse data files	173
Single constraints	185	export constraint values to	189
Single line comments	150	export variable values to	180
Single quotes, in names	149	Sparse data in models	
Singularity limit, CPLEX parameter	96	MPL Tutorial	309
SINH function	197	Sparse data vectors	30, 171
Size of models	144	Sparse Index and Data Handling	144
Skip over empty, Excel model file option	154	SPARSE keyword	171
Slack keyword	189, 227	SPARSEFILE keyword	173, 180, 189
Slack values		Speed of model generation	144
in solution file	83	Spreadsheet	
shown in a View window	64	Excel workbook file option	154
Solution file		Spreadsheet view of matrix	
contents	83	in view window	72
generate	82	SQL statements	
MPS names	82	send to Message Window	31, 70, 76
nonzero values only	82	SQR function	197
shown in a View window	62, 238	SQRT function	197
zero values as dot	82	Start algorithm, CPLEX parameter	102
Solution File Options dialog box	82	Start MPL	16
compute ranges	83	MPL Tutorial	234
decimals	82	Starting point algorithm, CPLEX parameter	109
filename	82	Status Bar, described	23
generate solution file	82	Status Window	58
nonzero values only	82	progress while solving models	26, 58, 237
number width	82	send messages to Message Window	31, 70, 76
solution file contents	83	Step-by-Step description	
solution filename	82	Tutorial Session 1, Model1.mpl	234
use MPS names	82	Tutorial Session 2, Bakery2.mpl	248
zero values as dot	82	Tutorial Session 3, Planning3.mpl	260
Solution info		Tutorial Session 4, Planning4.mpl	274
in solution file	83	Tutorial Session 5, Planning5.mpl	287
Solution mapping file, general solver option	86	Tutorial Session 6, Planning6.mpl	301
SOLUTIONFILE keyword	180, 189	Tutorial Session 7, Planning7.mpl	316
Solve current model	59	Structure of the MPL model file	145
Solve the model		Structured bounds	192
from the Run menu	58, 237	Structured constraints	155, 187
from the toolbar	58	Sub algorithm, CPLEX parameter	102
Solver info		Subindexes	159, 192, 199
in solution file	83	using in constraints	188
Solver iteration log		SUBJECT TO keyword	145, 148, 184, 185
send to Message Window	31, 70, 76	Subrange	156
Solver menu setup	58, 236	Subranges of indexes in constraints	188
Solver Menu Setup dialog box	132	Subscript arithmetic	214
Solver Options List dialog box	131	Subscript length	157, 165, 166
Solver Setup Options dialog box	133	Subscripted constraints	187
Solvers		Subscripted variables	155, 178
change setup	133	Subscripts	199
DOS legacy solvers	58, 62, 132	direct assignment of	215
DOS solver options	134	fixed	199
filename	133	Subset of an index	159
for MPL	132	SUM keyword	205
in the Run menu	58, 236	Summations	
select for the Run menu	11, 132	using in MPL	205
supported for MPL	132	Supported solvers for MPL	132
Solving, pause during	134	Syntax check the model	57
SOS keyword	148, 193	MPL Tutorial	249
SOS reference rows, generate MPS option	84	System info	

in solution file	83
System requirements, hardware	10

T

TAN function	197
TANH function	197
Terms in formulas	200
Text editor	
clipboard operations	46
close model file	39
copy text to the clipboard	46
create a new model file	37
cut text to the clipboard	46
delete text in the model file	46
goto line in the model	50
insert file into the editor	42
open model file	38
paste text from the clipboard	46
print file	43
replace text in the model	49
save model file	40
save selected text to a file	41
search for text in the model	48
undo changes	45
Textfile	144
THEN keyword	202
Tile windows	135
Time elapsed	
shown in Status Window	26
Time limit, CPLEX parameter	95
Time limit, XPRESS parameter	120
TITLE keyword	145, 147
Title of the model	147
Tolerance Options for XPRESS dialog box	122
Toolbar buttons	22
Edit Copy	46
Edit Cut	46
Edit Paste	46
File Open	38
File Print	43
File Save	40
Project New	52
Project Open	52
Project Save	54
Run Check Syntax	57
Run Solve	58
Search Find	48
Search Goto Line	50
Search Replace	49
View Message Window	70
View Model Definitions	66
View Solution File	62
Trace infeasibility, XPRESS parameter	117
Transportation models, MPL Tutorial	296
Tree memory limit, CPLEX parameter	96
Tutorial Concepts	
alias indexes	296
balance constraints	297

data constants	257
data files	311
data, variable, constraint vectors	256
equipment indexes	310
external data files	284
IN operator	310
index files	311
indexes as domains	256
inventory balance constraints	270
inventory balance constraints	271
period indexes	270
plant balance constraints	297
plant index	284
sales and inventory variables	270
transportation models	296
transshipment models	296
using summations over vectors	257
where conditions	296
Tutorial folder	235
Tutorial model file	
Session 3, Planning3.mpl	259
Session 4, Planning4.mpl	273
Session 5, Planning5.mpl	286
Session 6, Planning6.mpl	300
Session 7, Planning7.mpl	314
Tutorial, MPL	
Session 1	233
Session 2	243
Session 3	255
Session 4	269
Session 5	283
Session 6	295
Session 7	309

U

<i>UI bound entries</i> , MPS files	85
Underscore, in names	149
Undo changes in the model editor	45
UNION keyword	161
Upper bounds	191
Upper cutoff, CPLEX parameter	106
Use advance basis, general solver option	85
Username, model file option	154
Using projects to manage models	32

V

Variable count	
shown in Model Statistics dialog box	65
shown in status window	27
Variable selection, CPLEX parameter	99
Variable upper, CPLEX parameter	110
Variable values	
in solution file	83
shown in a View window	64
Variable vectors, MPL Tutorial	256
Variables	198

MPL Modeling System

binary	179, 193
bounds	191
description	179
fixed	191
free	192
initial values	179
inside parentheses	204
integer	179, 193
name	178
plain variables must be defined	77, 152
refer to in formulas	200
semicolon	178
semi-continuous	192
shown in Model Definitions window	30, 68
WHERE condition on	180, 296
VARIABLES keyword	178
vector multiplication	215
Vectors	
constraint	187, 257
data	170, 256
MPL Tutorial	256, 257
multiplication in formulas	200
summations	257
variable	178, 198
Vectors and indexes	
MPL Tutorial	255
View dense data vector elements	30
View menu	61
display model definitions window	66
display model statistics	65
display ranges objective	64
display ranges RHS	64
display slack/shadow prices	64
display solution files	62
display values/reduced cost	64
View message window	31, 70
from the toolbar	70
View Model Definitions window	28, 66
from the toolbar	66
MPL Tutorial	239
View named index elements	29, 67
View Other File dialog box	63
View solution file	62, 238
from the toolbar	62
View window	
display input file	62
display output file	62
display solution file	62, 238
Visual Basic, calling MPL from	17

W

Warning messages	
send to message window	31, 70, 76
WHERE conditions	210
database connection	166, 222
EXCEPT keyword used in	203
on constraints	188
on formula terms	202

on multi-dimensional index set	162
on variables	180
WHERE keyword	166, 180, 202, 222
White space, in MPL models	150
Window menu	135
arrange window icons	135
cascade windows	135
close all windows	135
list of open windows	135
tile windows	135
Windows application, calling MPL from	17
Windows, close all	135
Windows, list of open	135

X

XA Option Parameters dialog box	130
XA solver	
change options	130
XPRESS parameters	
Absolute gap	125
Algorithm	128
Barrier cache limit	121
Barrier iteration limit	121
Barrier memory limit	121
Barrier Options dialog box	128
Barrier progress info log	119
Barrier tolerance	123
Cholesky factorisation	129
Crash method	114
Cross-over control	128
Cut strategy	126
Cut-off	127
Degradation estimate	125
Dense column removal	129
Estimate deg. mult.	125
Extra processors for parallel MIP	121
Gomory cuts	127
Infeasibility sets	114
Initial solution	114
Level and frequency	127
Lifted cover inequalities	127
Limit Options dialog box	120
Log File Options dialog box	118
Logical preprocessing	117
LP iteration limit	120
LP iteration log	119
Markowitz tolerance	123
Max cuts	127
Max nonzero coeffs	127
MIP Cuts Options dialog box	126
MIP node log	119
MIP solutions limit	121
MIP Strategy Options dialog box	124
Node limit	121
Node selection	125
Nodeset selection	124
Numerical tolerance	122
Ordering algorithm	129

Part. pricing cand. list sizing	120	Trace infeasibility.....	117
Perturbation.....	114	Zero tolerance.....	123
Preprocessing Options dialog box.....	116		
Presolve	116		
Pricing.....	114		
Probing	117		
Pseudo cost	125		
Reduced cost fixing.....	117		
Relative gap	125		
Scaling	115		
Send iteration log to	118		
Simplex Options dialog box	113		
Tolerance Options dialog box	122		

Z	
Zero tolerance, XPRESS parameter	123
Zero values as dot, in the solution file	82
Zooming	
matrix graph window	72
objective function graph window.....	72