

# **ILOG OPL Studio 3.7**

## **Studio User's Manual**

**September 2003**

*Copyright © 1987-2003, by ILOG S.A. All rights reserved.*

*ILOG, the ILOG design, CPLEX, and all other logos and product and service names of ILOG are registered trademarks or trademarks of ILOG in France, the U.S. and/or other countries.*

*Java™ and all Java-based marks are either trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.  
Microsoft, Windows, and Windows NT are either trademarks or registered trademarks of Microsoft Corporation in the U.S. and other countries.*

*All other brand, product and company names are trademarks or registered trademarks of their respective holders.*



# Contents

<b>Preface</b>	<b>Before You Begin . . . . .</b>	<b>11</b>
	<b>About ILOG OPL Studio . . . . .</b>	<b>11</b>
	<b>What You Need to Know . . . . .</b>	<b>12</b>
	<b>What This Manual Contains . . . . .</b>	<b>12</b>
	<b>Notation Used in This Manual . . . . .</b>	<b>13</b>
	<b>Related Documentation . . . . .</b>	<b>13</b>
	<b>Where to Get More Information . . . . .</b>	<b>14</b>
	Users' Mailing List . . . . .	15
	Web Site. . . . .	15
	<b>Licensing Requirements . . . . .</b>	<b>15</b>
<b>Chapter 1</b>	<b>Overview of ILOG OPL Studio . . . . .</b>	<b>17</b>
	<b>Launching ILOG OPL Studio . . . . .</b>	<b>18</b>
	Launching an OPL Script in Batch Mode . . . . .	19
	Batch Mode for OPL Models . . . . .	19
	Other Command Line Options . . . . .	19
	Japanese Localization . . . . .	20
	<b>ILOG OPL Studio Main Window . . . . .</b>	<b>22</b>
	Menu Bar Commands . . . . .	25

Tool Bar Buttons .....	31
Execution Tool Bar Buttons .....	32
Dockable GUI Elements .....	33
<b>ILOG OPL Studio Basics .....</b>	<b>35</b>
File Types .....	35
Opening an Existing File .....	36
Creating a New File .....	36
Executing a Project or Model .....	38
Checking for Syntactic or Semantic Errors .....	39
Specifying Processing Directives .....	40
Terminating ILOG OPL Studio .....	40
<b>The Text Editor .....</b>	<b>41</b>
Switching Between Editor Windows .....	41
Resizing an Editor Window .....	42
Editor Quick Reference .....	43
Customizing the Editor .....	46
<b>The Online Help .....</b>	<b>48</b>
Windows Platforms .....	48
UNIX Platforms .....	49
<b>Chapter 2 Tutorial: Working with Projects .....</b>	<b>51</b>
<b>The Production Planning Example .....</b>	<b>52</b>
<b>Creating a Project .....</b>	<b>54</b>
Inserting an Existing Model File into a Project .....	55
Inserting an Existing Data File into a Project .....	57
Adding New Files to a Project .....	58
Saving the Project .....	59
Setting Project Options .....	60
<b>Executing the Project .....</b>	<b>62</b>
<b>Examining a Solution to the Model .....</b>	<b>63</b>
Using the Output Area .....	63
Using the Model Browser .....	67

	<b>Continuing the Execution</b> .....	<b>71</b>
	<b>Using an Alternative Data File</b> .....	<b>73</b>
	<b>Closing a Project File</b> .....	<b>74</b>
	<b>Working with Several Projects</b> .....	<b>74</b>
<b>Chapter 3</b>	<b>Tutorial: Predefined Dynamic Display</b> .....	<b>79</b>
	<b>The Car Sequencing Example</b> .....	<b>80</b>
	<b>Setting Up the Project</b> .....	<b>83</b>
	Opening the Project File .....	83
	Loading the Data .....	83
	<b>Executing the Project</b> .....	<b>85</b>
	<b>Examining the First Solution</b> .....	<b>86</b>
	<b>Copying the Results Matrix to a Spreadsheet</b> .....	<b>88</b>
	<b>Continuing the Execution</b> .....	<b>88</b>
	<b>Looking at the Model Structure</b> .....	<b>89</b>
	<b>Using Dynamic Display with ILOG OPL Studio</b> .....	<b>90</b>
	<b>Closing the Project</b> .....	<b>92</b>
<b>Chapter 4</b>	<b>Tutorial: Examining the Solution to a Scheduling Problem</b> .....	<b>93</b>
	<b>The House Building Example</b> .....	<b>94</b>
	<b>Opening the Model File</b> .....	<b>95</b>
	<b>Looking at the Model Structure</b> .....	<b>96</b>
	<b>Executing the Model</b> .....	<b>98</b>
	<b>Examining the Solution</b> .....	<b>100</b>
	Looking at the Activities Results .....	100
	Looking at the Resources Results .....	107
	The Optimization Notebook Page .....	109
	The Solver Notebook Page .....	109
	<b>Completing the Execution</b> .....	<b>110</b>
	<b>Closing the Model File</b> .....	<b>111</b>

<b>Chapter 5</b>	<b>Tutorial: Scheduling-Specific Dynamic Display</b>	<b>113</b>
	The Activity Domains Window	114
	The Bridge Example.	116
<b>Chapter 6</b>	<b>Tutorial: User-Defined Dynamic Display</b>	<b>119</b>
	The Drawing Board	119
	The Square Example	120
	The Map Example.	122
	The Euler Example.	122
<b>Chapter 7</b>	<b>Tutorial: Debugging the Search Strategy</b>	<b>123</b>
	A Basic Example with the Eight Queens Problem	124
	Setting up the Example	124
	Executing the Model.	124
	Continuing the Execution	127
	<b>The Frequency Allocation Example</b>	<b>130</b>
	Looking at the Model Structure	135
	Setting the Debug Option.	136
	<b>Executing the Frequency Allocation Project.</b>	<b>137</b>
	Displaying the Stack Window	139
	Displaying the Inspector Window	139
	Continuing the Execution	140
	<b>Executing the Project with the ‘Stepping in Model’ Option</b>	<b>142</b>
	Displaying the Stack Window and Inspector Window	144
	Visualizing the Search Strategy with the Stack and Inspector Windows	147
	<b>Visualizing the Search Tree</b>	<b>148</b>
	Using the Depth First Search.	148
	Using the Slice-Based Search	150
	<b>Cooperating Solvers – Combined LP and CP</b>	<b>153</b>
	<b>Exploration Strategy – Drawing Board Combined with Search Tree</b>	<b>160</b>
	<b>Terminating ILOG OPL Studio</b>	<b>162</b>

<b>Chapter 8</b>	<b>Customizing ILOG OPL Studio</b>	<b>163</b>
	<b>Default Options and Project Options</b>	<b>164</b>
	Setting the Default Options	164
	Setting Project Options	164
	Navigating in the Options Dialog Boxes	164
	<b>Setting Constraint Programming Options</b>	<b>165</b>
	<b>Setting Editor Options</b>	<b>166</b>
	Changing the Fonts	167
	Changing the Foreground and Background Colors	168
	<b>Setting Output Options</b>	<b>169</b>
	<b>Setting Advanced Options</b>	<b>171</b>
	<b>Setting Miscellaneous Options</b>	<b>172</b>
<b>Chapter 9</b>	<b>Mathematical Programming</b>	<b>175</b>
	<b>MP General</b>	<b>176</b>
	<b>Optimization Using Simplex</b>	<b>180</b>
	<b>Preprocessing</b>	<b>185</b>
	<b>Mixed Integer Programming</b>	<b>189</b>
	MIP Strategy	189
	MIP Limits	194
	MIP Tolerances	198
	MIP Cuts	201
	<b>Barrier Algorithm</b>	<b>205</b>
	<b>Network Simplex Algorithm</b>	<b>209</b>
	<b>Results of Mathematical Programming</b>	<b>211</b>
<b>Chapter 10</b>	<b>Working with a Database</b>	<b>213</b>
	<b>Supported Databases</b>	<b>214</b>
	<b>Database Connectivity</b>	<b>214</b>
	<b>Prerequisites</b>	<b>216</b>
	<b>The Bridge Example</b>	<b>216</b>
	<b>Setting Up the Database</b>	<b>217</b>

	<b>The Data Tables .....</b>	<b>218</b>
	The Task Table .....	219
	The Resource Table.....	220
	The MAX and MIN Tables .....	220
	The Precedence Table.....	221
	The Requirements Table .....	221
	<b>The OPL Model.....</b>	<b>222</b>
	Record Definitions .....	222
	Connecting to the Database from OPL .....	223
	Reading From the Database .....	224
	Creating a New Table and Updating the Database .....	225
	<b>Executing the Bridge Example .....</b>	<b>226</b>
	Viewing the Result in the Database.....	226
	Consulting the Result From Another Model.....	229
<b>Chapter 11</b>	<b>Using OPLScript.....</b>	<b>231</b>
	<b>The Vellino Example .....</b>	<b>232</b>
	<b>Opening the Script File .....</b>	<b>233</b>
	<b>Executing the Script .....</b>	<b>234</b>
	<b>Stepping in a Script.....</b>	<b>236</b>
	To Step to the First Instruction.....	236
	To Step to the Next Instruction.....	237
	To Abort the Execution.....	237
	To Deselect the Stepping Option .....	237
	To Continue without Stepping .....	237
	To Step Out of a Loop .....	237
	To Complete the Execution .....	239
	<b>Adding and Removing Breakpoints.....</b>	<b>240</b>
	<b>Closing the Script File.....</b>	<b>240</b>
<b>Chapter 12</b>	<b>Generating Compiled Models .....</b>	<b>241</b>



**Appendix A    OPL Parameters ..... 243**

**List of Figures ..... 255**

**List of Tables ..... 259**

**List of Code Samples..... 261**

**Index .....263**



## ***Before You Begin***

This manual provides you with all the information you need for installing and using ILOG OPL Studio. It contains several tutorials with examples and step by step explanations.

---

### **About ILOG OPL Studio**

ILOG OPL Studio is an integrated development environment for mathematical programming and combinatorial optimization applications. It is the graphic user interface for the OPL modeling language. All development effort is supported through the various modules accessible via ILOG OPL Studio.

With ILOG OPL Studio, you can:

- ◆ create and modify model and project files using the editing capabilities
- ◆ execute a model or project
- ◆ debug OPL statements using the debug facilities
- ◆ visualize OPL results, using 2D graphic representation or textual representation
- ◆ dynamically visualize the state of variables during the search for a solution
- ◆ display the constraint programming search tree
- ◆ select mathematical programming options
- ◆ connect to a database

- ◆ work with OPLScript, the OPL scripting language
- ◆ generate a compiled model

---

## What You Need to Know

This manual assumes that you are familiar with the UNIX or PC environment in which you are going to use ILOG OPL Studio, including its particular windowing system.

You should also be familiar with the OPL modeling language. The tutorial examples referred to in this manual are taken from the *ILOG OPL Studio: Language Manual*.

In the present manual, the programming and language concepts behind the examples are not explained. You will need to refer to the *ILOG OPL Studio: Language Manual* for an explanation of the modeling language of these examples. The present manual describes only how to use the features of ILOG OPL Studio.

---

## What This Manual Contains

- ◆ **Chapter 1, *Overview of ILOG OPL Studio***, describes the basics of ILOG OPL Studio – how to launch the product, the Main window and its elements, the text editor, the keyword help.
- ◆ **Chapter 2, *Tutorial: Working with Projects***. A simple production planning example introduces you to ILOG OPL Studio.
- ◆ **Chapter 3, *Tutorial: Predefined Dynamic Display***. The car sequencing example contains a constraint model showing constraint programming constructs.
- ◆ **Chapter 4, *Tutorial: Examining the Solution to a Scheduling Problem***. The house building example illustrates how ILOG OPL Studio can handle a scheduling problem.
- ◆ **Chapter 5, *Tutorial: Scheduling-Specific Dynamic Display***. The bridge example shows how to define a scheduling-specific dynamic display and use the Activity Domains window.
- ◆ **Chapter 6, *Tutorial: User-Defined Dynamic Display***. The examples presented use the Drawing Board to animate the search algorithm.
- ◆ **Chapter 7, *Tutorial: Debugging the Search Strategy***. The frequency allocation example demonstrates the debugging facilities of ILOG OPL Studio, including the search tree.
- ◆ **Chapter 8, *Customizing ILOG OPL Studio***, provides information about setting ILOG OPL Studio options to meet your particular needs.

- ◆ **Chapter 9, *Mathematical Programming***, describes the CPLEX options available in ILOG OPL Studio.
- ◆ **Chapter 10, *Working with a Database***, illustrates how to connect to a database from ILOG OPL Studio, and how to read from and write to the database.
- ◆ **Chapter 11, *Using OPLScript***, describes how to work with the OPL scripting language.
- ◆ **Chapter 12, *Generating Compiled Models***, explains how to generate a .opl file that can be integrated into your application.
- ◆ **Appendix A, *OPL Parameters***, provides an alphabetical list of OPL parameters with their values.

---

## Notation Used in This Manual

The following typographic conventions apply throughout this manual:

- ◆ code extracts and file names are written in `this typeface`.
- ◆ entries to be made by the user are written <in angle brackets>
- ◆ commands appear as: File>Open.

---

## Related Documentation

- ◆ *ILOG OPL Studio: Language Manual*  
Provides a description of the OPL and OPLScript programming languages used in our examples and should be read in conjunction with the present document.
- ◆ An online help, accessible from OPL Studio, contains a quick reference to the OPL and OPLScript languages.
- ◆ *ILOG OPL Studio: Component Libraries Reference Manual*  
Describes the various APIs available for accessing the OPL solving engine, including the Microsoft COM/ActiveX API, a native C++ API, and a Java API. Microsoft's Component Object Model (COM) allows Windows users to access OPL from within languages such as Visual Basic.
- ◆ *ILOG OPL Studio: Component Libraries User's Manual*  
Contains examples that show how to use the APIs in order to access OPL from Excel VBA, Visual Basic, C++, and Java. Examples of integration into web servers with ASP and JSP are also provided.

◆ *ILOG OPL Studio: Release Notes*

Indicate the new and modified features of each release.

◆ Source code for examples delivered in the standard distribution.

◆ A `readme.txt` file delivered as part of the standard distribution. This file contains the most up-to-date information about platform prerequisites for ILOG OPL Studio.

---

## Where to Get More Information

For technical support of OPL Studio, contact your local distributor, or, if you are a direct ILOG customer, contact:

Region	E-mail	Telephone	Fax
France	oplstudio-support@ilog.fr	0 800 09 27 91 (numéro vert) +33 (0)1 49 08 35 62	+33 (0)1 49 08 35 10
Germany	oplstudio-support@ilog.de	+49 6172 40 60 33	+49 6172 40 60 10
Spain	oplstudio-support@ilog.es	+34 91 710 2480	+34 91 372 9976
United Kingdom	oplstudio-support@ilog.co.uk	+44 (0)1344 661 630	+44 (0)1344 661 601
Rest of Europe	oplstudio-support@ilog.fr	+33 (0)1 49 08 35 62	+33 (0)1 49 08 35 10
Japan	oplstudio-support@ilog.co.jp	+81 3 5211 5770	+81 3 5211 5771
Singapore	oplstudio-support @ilog.com.sg	+65 6773 06 26	+65 6773 04 39
North America	oplstudio-support@ilog.com	1-877-ILOG-TECH 1-877-8456-4832 (toll free) or 1-650-567-8080	+1 650 567 8001

We encourage you to use e-mail for faster, better service.

---

## Users' Mailing List

The electronic mailing list `oplstudio-list@ilog.fr` is available for you to share your development experience with other OPL users. This list is not moderated, but subscription is subject to an on-going maintenance contract. To subscribe to `oplstudio-list`, send an e-mail without any subject to `oplstudio-list-owner@ilog.fr`, with the following contents:

```
subscribe oplstudio-list
your e-mail address if different from the From field
first name, last name
your location (company and country)
maintenance contract number
maintenance contract owner's last name
```

---

## Web Site

On our web sites, you will find a wealth of information about constraint programming in a range of articles and conference papers explaining the theoretical background and technical features of OPL Studio and other ILOG products.

In addition to those freely accessible pages, there are also technical support pages on our web sites. They contain FAQ (Frequently Asked/Answered Questions) and the latest patches for some of our products. Changes are posted in the product mailing list. Access to these pages is restricted to owners of an on-going maintenance contract. The maintenance contract number and the name of the person this contract is sent to in your company will be needed for access, as explained on the login page.

All three of these sites contain the same information, but access is localized, so we recommend that you connect to the site corresponding to your location, and select the “Tech Support Web” page from the home page.

Americas:	<a href="http://www.ilog.com">http://www.ilog.com</a>
Asia and Pacific Nations:	<a href="http://www.ilog.com.sg">http://www.ilog.com.sg</a>
Europe, Africa, and Middle East:	<a href="http://www.ilog.fr">http://www.ilog.fr</a>

---

## Licensing Requirements

- ◆ To use the OPL Studio 3.7 graphic environment you need the key OPLStudio 3.
- ◆ To use the free trial version of OPL Studio 3.7, which has restrictions on model size and does not allow the compiling of models for deployment, you do not need a key. The evaluation period is 6 months. The free trial version does not include the OPL Component Library, and not all database drivers are provided.

- ◆ To use OPL Studio 3.7 in batch mode you need the keys for the underlying libraries, as required by your model (CPLEX and/or Solver and/or Scheduler).
- ◆ The keys required to execute an OPL model, or a compiled OPL model, using the OPL Component Libraries are indicated in the Preface of the *ILOG OPL Studio: Component Libraries User's Manual*.

Information on licensing requirements can also be found in the file `readme.txt`, delivered with OPL Studio.



## ***Overview of ILOG OPL Studio***

This chapter describes the basic features of ILOG OPL Studio:

- ◆ how to launch the product
- ◆ the main window and its elements
- ◆ the basic concepts for using ILOG OPL Studio
- ◆ the text editor
- ◆ the online help

## Launching ILOG OPL Studio

In this manual we assume that you have already successfully installed ILOG OPL Studio on your particular platform. If this is not the case, refer to the booklet delivered with the OPL Studio CD-ROM.

Once you have installed the product, you are ready to launch ILOG OPL Studio.

### ◆ For Windows XP, Windows 2000, Windows NT 4, Windows 98

Click the Start menu and then select:

```
Programs>Ilog>ILOG OPL Studio 3.7>OPL Studio 3.7
```

#### In Batch Mode

Use the option `-batch` in the command line.

### ◆ For UNIX Systems

Enter:

```
oplst
```

On HP systems you must set `SHLIB_PATH` instead of `LD_LIBRARY_PATH`. You need to add the shared libraries to the `env` variable:

```
set env SHLIB_PATH=/usr/ilog/OPLSt37/lib/hp32_11_3.15/shared
```

#### In Batch Mode

Use the option `-batch` in the command line.

A **configuration file**, `oplst3.config`, is created in each directory from which you launch OPL Studio. When you quit the application, the configuration options (main window setup, dockable pane positions) are saved in this file. Each time you relaunch the GUI, these options are restored and the GUI has the same appearance as the last time you used it.

When using OPL Studio in batch mode you need licenses for the underlying libraries as required by your model (Solver, Scheduler and/or CPLEX).

`opl.bat` launches `oplst.exe` with the `-batch` option and redirects the standard output to a file called `result.txt`. Then it displays this file's content and deletes it. Because OPL Studio is a Windows application and not a console application, you would not be able to see the output if you launched directly `oplst.exe -batch` without redirecting the standard output. This is why we provide the `opl.bat` file.

When using the batch mode, you must avoid interactive instructions in your model such as:

```
int n << "number of queens:";
```

which is used in the model `queens.mod` and displays a window requiring input from the user.

---

## Launching an OPL Script in Batch Mode

### On UNIX

```
oplst -batch ../../examples/opl/scripts/gomory.osc
```

### On Windows

A .bat file is available in the bin directory:

```
opl.bat ..\examples\opl\scripts\gomory.osc
```

---

## Batch Mode for OPL Models

### On UNIX

```
oplst -batch ../../examples/opl/gas.mod ../../examples/opl/gas.dat
```

### On Windows

```
opl.bat ..\examples\opl\gas.mod ..\examples\opl\gas.dat
```

---

## Other Command Line Options

### Path Option

```
-path <include_path>
```

The `-path` option specifies a directory in which an OPL script can find the OPL model files and include files.

### Help Option

```
-help
```

The `-help` option displays a message describing the documented options available in command line mode, then exits.

### Version Option

```
-version
```

The `-version` option displays a message with the version number and build date of the executable, then exits.

## Japanese Localization

If you run OPL Studio on a Windows platform with the Japanese regional settings, you will see labels in Japanese. There is no need to install another executable or a message database.



These labels can be displayed in English by relaunching the application and adding a startup parameter.

## Switching from Japanese to English

If you want to switch to the English (US) version, you need to relaunch OPL Studio with the parameter `-us`. There are two ways of doing this.

- ◆ In MS-DOS, enter the command:

```
C:\ILOG\OPLSt37\bin\oplst.exe -us
```

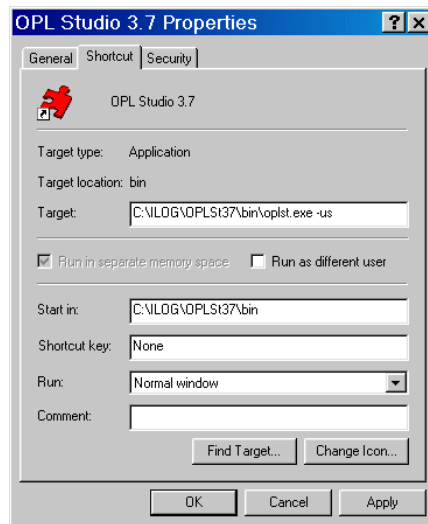
- ◆ On your Windows desktop, right-click on the OPL Studio shortcut and select Properties. In the dialog box (see Figure 1.1), do the following:

- Select the Shortcut tab
- In the Target field, add the parameter `-us`

This field should now contain

```
C:\ILOG\OPLSt37\bin\oplst.exe -us
```

- Click OK.



**Figure 1.1** Switching from the Japanese Version to the English (US) Version

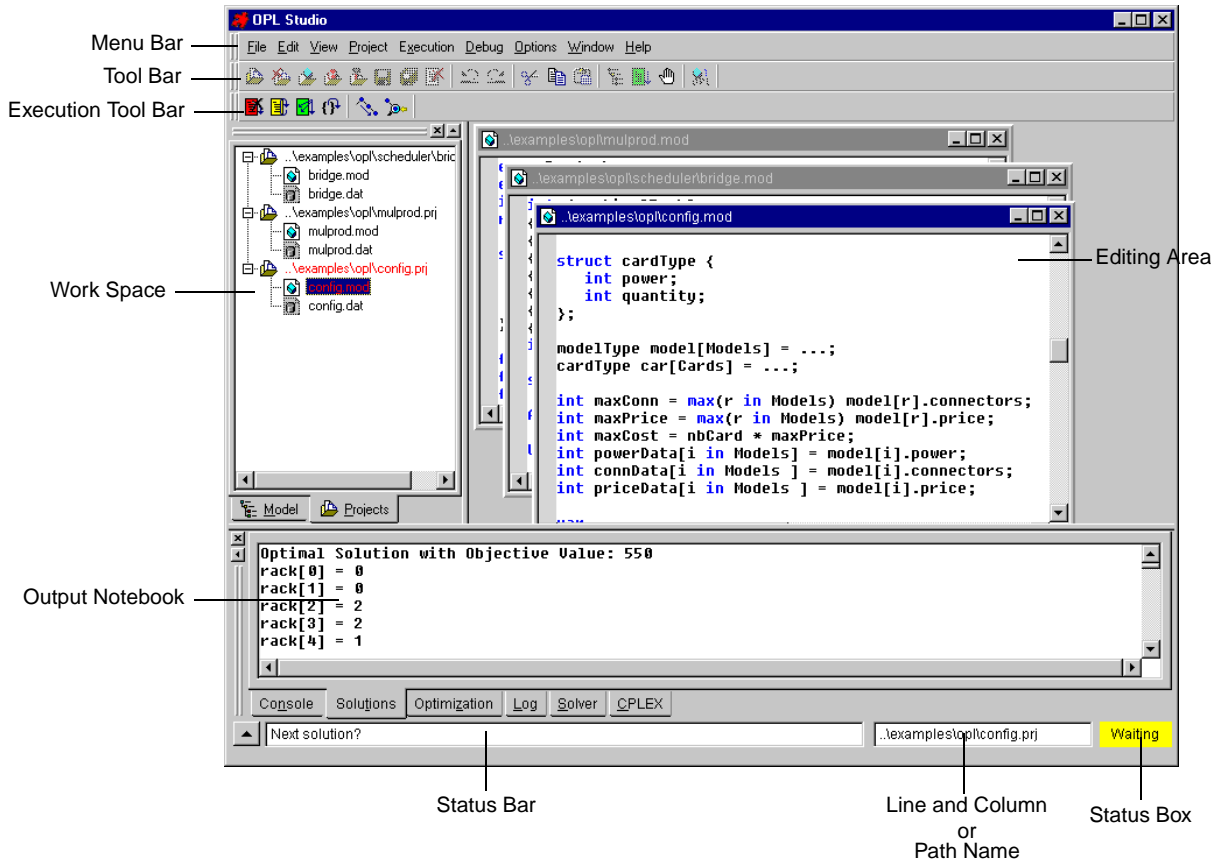
## Reading from a Database

OPL cannot read a double-byte string from a database. This means that all string data from a database must contain standard single-byte characters.

This applies whether the regional settings are Japanese or English.

## ILOG OPL Studio Main Window

When you launch ILOG OPL Studio, the Main window appears. All tasks and commands for using ILOG OPL Studio are carried out from this window. The figure below shows the Main window with three projects open.




**Figure 1.2** ILOG OPL Studio Main Window

**Note:** If you have a mouse with a wheel between the two buttons, you can use the wheel to scroll up and down.

- ◆ **Menu Bar** Use these menus to choose various commands, such as Save in the File menu, and to display dialog boxes to perform various tasks. Certain menu commands have their own buttons just below the menu bar.
- ◆ **Tool Bar** These buttons are provided for frequently-used commands.
- ◆ **Execution Tool Bar** Contains the buttons for commands used during execution mode.
- ◆ **Work Space** Contains a notebook with two pages:

**Projects** More than one project can be open at the same time. This page displays project tree structures containing all the files related to each project. It also displays stand-alone models and scripts.

**Model** This page displays the model browser, containing information about the data structures defined in the active model. From the model browser you can select display options for the variables before executing a model, and open visualization windows after execution. The active model is browsed using one of the following methods:

- Click the "Rebuild Browser Information" button  in the tool bar
  - Select "Browse Active Model" from the Execution menu
  - Select "Browse Active Model" from the model browser root contextual menu, accessible through right-clicking on the root item
  - Select "Browse Model" from the context-sensitive menu, accessible through right-clicking on the active project in the Projects notebook page.
- ◆ **Editing Area** This area displays opened model, data, script, or C++ files. Use this area to create new files, edit existing files, or examine active documents. You can open more than one file in this space. The opened files are displayed in separate panels with the file name appearing in the title bar.
  - ◆ **Output Notebook** This area is used by ILOG OPL Studio to return error information, solutions, and results. Separate notebook pages appear for console messages, solutions to models, optimization information, the log of past actions, solving statistics, and additional pages appear for other displays depending on the nature of the model.
  - ◆ **Status Bar** This area displays messages concerning the execution status of ILOG OPL Studio. These messages are then stored in the Log notebook page.
  - ◆ **Path Name or Line and Column** This area displays the pathname of the file just loaded, or the file being executed. When moving the insertion point in the editor, this area displays the line number and column number.

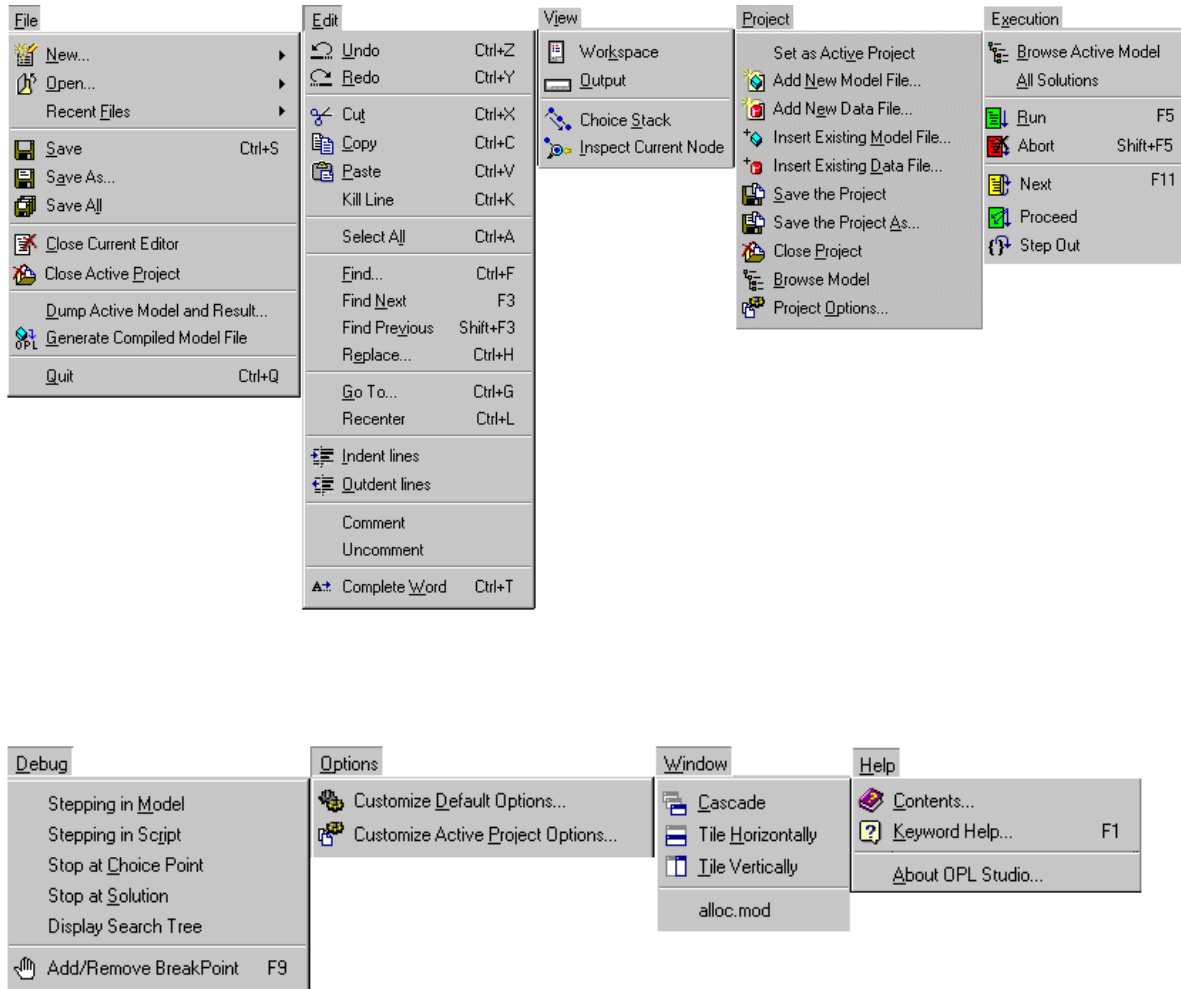
- ◆ **Status Box** This box changes its color and label according to the status of the currently active model, project or script:

Color	Label	Meaning
Blue	Idle	Indicates that ILOG OPL Studio is idle. Files can be created, edited or examined in the editing area.
Green	Running	Indicates that ILOG OPL Studio is executing the active model, project or script.
Yellow (flashing)	Waiting	Indicates that ILOG OPL Studio is in a waiting state during execution mode. It is waiting for further direction from the user to continue or stop the execution.
Red	Aborting	Indicates that ILOG OPL Studio is aborting the execution.

The default behavior of the yellow box is to blink in order to remind you that OPL Studio is in a waiting state. You can stop this effect in the Advanced page of the Default Options dialog box. See *Setting Advanced Options* on page 171.



## Menu Bar Commands

**Figure 1.3** Overview of Commands in the Menu Bar

Some of the menu items have a keyboard shortcut, indicated in the right-hand column of the menu. For example, Keyword Help has the shortcut F1, which means that you can obtain help on an OPL keyword in the text editor by selecting the keyword and then either clicking the item Keyword Help, or pressing the key F1.

The table below lists the commands found in the menus and provides a description of each command.

**Table 1.1** *Commands in the Menu Bar*

Command	Description
<b>File</b>	
New	Creates a new file. Displays a submenu to specify a model, project, data or script file.
Open	Opens a file. Displays a submenu to specify a model, project, data, script, or C++ file.
Recent Files	Files that you have used recently can be opened by selecting the file names from the submenu.
Save	Saves the current edited file.
Save As	Saves the current edited file with a new name.
Save All	Saves all the open files.
Close Current Editor	Closes the current edited file.
Close Active Project	Closes the active project.
Dump Active Model and Result	Allows you to save current ILOG OPL Studio information to a file. OPL Studio saves all the model and data files for the active project, anything in the Solutions and Optimization windows, and the current user settings for solving the problem.
Generate Compiled Model File	Generates a compiled OPL model (.opl file) from the active model.
Quit	Exits ILOG OPL Studio.
<b>Edit</b>	
Undo	Undoes an unlimited number of nested actions in the current editor.
Redo	Redoes previously undone actions in the current editor (unlimited).

**Table 1.1** *Commands in the Menu Bar (Continued)*

Command	Description
Cut	Deletes the selected text from the editor and puts it in the clipboard.
Copy	Copies the selected text, from the editor or output window, to the clipboard.
Paste	Pastes from the clipboard to the current editor.
Kill Line	Deletes a line from the cursor position onward and appends it to the clipboard.
Select All	Selects the entire content of the current editor.
Find	Displays the Find dialog box for specifying search criteria.
Find Next	Finds the next occurrence of the text displayed in the Find box.
Find Previous	Finds the previous occurrence of the text displayed in the Find box.
Replace	Displays the Replace dialog box for specifying search criteria and replacing specified strings.
Go To	Displays the Go To dialog box for specifying a line where the cursor should be placed in the work space.
Recenter	Places the current line in the middle of the window, if possible.
Indent Lines	Adds a tabulation at the beginning of selected lines.
Outdent Lines	Removes a tabulation from the beginning of selected lines.
Comment	Transforms selected lines to comments.
Uncomment	Uncomments selected lines.
Complete Word	Searches upward for a string in the same file and completes with the first string that matches the same beginning letters. The search begins upward at the left of the current cursor position.
<b>View</b>	
Workspace	Displays the Workspace notebook containing the model browser and the project tree.

**Table 1.1** *Commands in the Menu Bar (Continued)*

Command	Description
Output	Displays the Output area.
Choice Stack	Displays the Stack dialog box.
Inspect Current Node	Displays the Inspect dialog box.
<b>Project</b>	(This menu depends on the selection made in the Project Tree.)
Set as Active Project	When several projects are open, remembers the project selected in the Project Tree as the active one.
Add New Model File	Adds a newly-created model file to the project.
Add New Data File	Adds a newly-created data file to the project.
Insert Existing Model File	Adds an existing model file to the project.
Insert Existing Data File	Adds an existing data file to the project.
Save the Project	Saves the project's options and its components.
Save the Project As	Changes the name of an existing project.
Close Project	Closes the project.
Browse Model	Builds or rebuilds the model tree of the data structures defined in the active model or project.
Project Options	Opens the Project Options dialog box for the project.
<b>Execution</b>	
Browse Active Model	Builds or rebuilds the model tree of the data structures defined in the active model or project.
All Solutions	When checked, solves for all solutions during the execution. Does not enter a waiting state after each solution.
Run	Executes the active model, project or script.
Abort	Stops execution of a model, project or script and returns to a normal editing session. After pressing the Abort button, the traces of any solutions found up to that point are kept in the Output notebook.

**Table 1.1** *Commands in the Menu Bar (Continued)*

Command	Description
Next	Goes to the next solution of a model or project, or next choice point or next instruction.
Continue Run	Forces OPL Studio to produce all the remaining solutions.
Step Out	Avoids going through all the iterations of a loop when executing a script.
<b>Debug</b>	
Stepping in Model	During an execution, forces ILOG OPL Studio to stop at each instruction in a search procedure.
Stepping in Script	During an execution, forces ILOG OPL Studio to stop at each instruction in the script.
Stop at Choice Point	During an execution, forces ILOG OPL Studio to stop at each choice point.
Stop at Solution	During an execution, forces ILOG OPL Studio to stop at each solution in an optimization.
Display Search Tree	During the execution of a constraint programming model, displays the corresponding search tree.
Add/Remove Break Point	Used to set (and remove) breakpoints in the search procedure of an OPL model or script.
<b>Options</b>	
Customize Default Options	Displays the Default Options dialog box that allows you to change solver, font, color and graphics options.
Customize Active Project Options	Displays the Project Options dialog box that allows you to change solver, font, color and graphics options for the active project.
<b>Window</b>	
Cascade	Displays overlapping panels in the editing area.
Tile Horizontally	Displays panels in the editing area horizontally.
Tile Vertically	Displays panels in the editing area vertically.

**Table 1.1** *Commands in the Menu Bar (Continued)*

Command	Description
<b>Help</b>	
Contents	Opens the Help window, displaying the presentation page.
Keyword Help	Opens the Help window, displaying the page corresponding to the OPL keyword selected in the text editor.
About OPL Studio	Indicates the version of ILOG OPL Studio, the ILOG products used by OPL Studio, and contains copyright information.

## Tool Bar Buttons

The following buttons appear in the tool bar:



- **Load Project File**

Opens a project. ILOG OPL Studio displays an Open File dialog box requesting the file name of the project you wish to open. The Project dialog box then appears.



- **Close Active Project**

Saves and closes the active project file (.prj).



- **Load Model File**

Opens a model file (.mod) in the editing area.



- **Load Data File**

Opens a data file (.dat) in the editing area.



- **Load Script File**

Opens a script file (.osc) in the editing area.



- **Save the Editor Content**

Saves the current file in the editing area.



- **Save All Files**

Saves all the files in the editing area.



- **Close Current Editor**

Closes the active file in the editing area.



- **Undo**

Undoes your modifications to an edited file without a limit.



- **Redo**

Redoes what you have just undone, without a limit.



- **Cut**

Cuts selected text.



- **Copy**

Copies selected text to the clipboard.



- **Paste**

Pastes text from the clipboard.



- **Rebuild Browser Information**

Builds or rebuilds the model tree of the data structures defined in the active model or project, and displays the Model Browser notebook page in the work space.



- **Run**

Submits the active model, project or script for execution.



- **Add/Remove Breakpoint**

Sets (and removes) breakpoints in the search procedure of an OPL model or script.



- **Generate Compiled Model File**

Generates a compiled OPL model file (.opl).

---

## Execution Tool Bar Buttons

The execution tool bar appears after clicking the Run button.

- **Abort**



Stops the current computation during execution of the model, project or script. After an Abort, the traces of any solutions found up to that point are kept in the Output notebook.

- **Next**



Goes to the next solution of the model or project, or to the next instruction in stepping mode, or to the next choice point in 'stop at choice point' mode.

- **Continue Run**



Forces ILOG OPL Studio to produce all the remaining solutions without further intervention.

- **Step Out**



Steps out of a loop in a script to avoid going through all the iterations.

- **View Choice Stack**



Inspects the entire execution stack.

- **Inspect Current Node**



Inspects the current choice point in the execution.



## Dockable GUI Elements

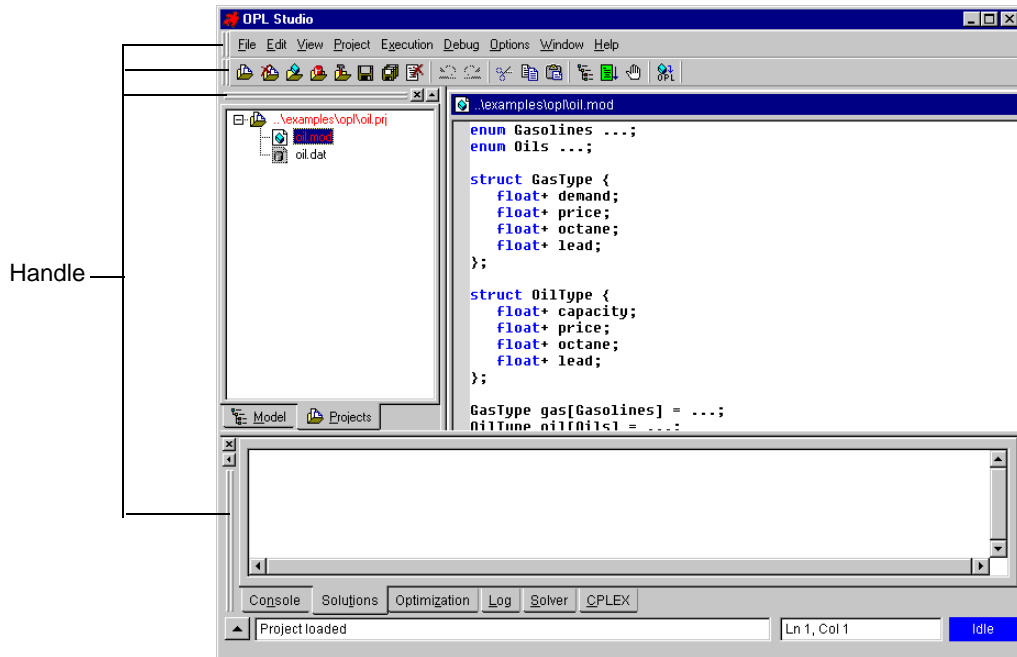
In the graphic interface the following elements are dockable:

Menu Bar	Output Area	Stack window
Tool Bar	Drawing Board	Inspector window
Work Space	Activity Domains Window	Search Tree

A dockable element can be detached from, or floated in, its own frame window or it can be attached to, or docked at, any side of its parent window.

## Floating a Window

To float a window, drag it outside the Main Window by its handle, materialized by double horizontal or vertical lines, as shown in Figure 1.4. When the ghost frame thickens, drop the window and let it float.



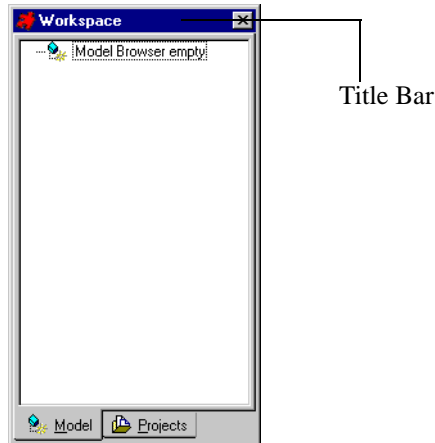
**Figure 1.4** Handles on Dockable Windows

You can alternatively double-click on the handle to float a window. In order to float the tool bar or menu bar, double-click on the background.

## Docking a Window

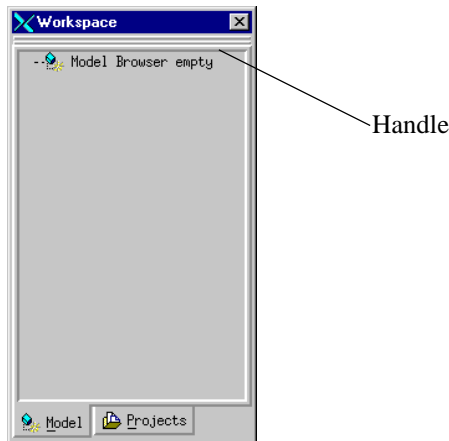
To dock a window, drag it back inside the Main Window and drop it.

In Microsoft Windows, you must drag and drop the title bar (see Figure 1.5). Alternatively, double-click on the title bar.




**Figure 1.5** Title Bar on Floating Window (Windows XP, 2000, NT 4, 98)

In UNIX, you must drag the handle inside the window, as shown in Figure 1.6. Dragging the title bar will have no effect on UNIX platforms (X-Windows).



**Figure 1.6** Handle on Floating UNIX Window (X-Windows)

## Hiding a Docked Window

To hide a docked window, just click on the button with a cross in it .

To display it again, choose the corresponding window in the View Menu.

## ILOG OPL Studio Basics

This section describes several basic concepts to consider when you use ILOG OPL Studio.

### File Types

#### ◆ Models

Model files contain OPL statements. A stand-alone model is a model that can be executed in OPL Studio without any additional requirements. A model file can be generated in a compiled form for integration into the OPL component libraries.

#### ◆ Data files

Large problems are better organized by separating the model of the problem from the instance data. The instance data is stored in a data file (or in several data files).

#### ◆ Projects

ILOG OPL Studio uses the concept of a project to associate a model file with a number of data files. The model file declares the data but does not initialize it. The data files contain the initialization of each data item declared in the model. The project file then organizes all the related model and data files. A project provides a convenient way to maintain the relationship between related files and runtime options for the environment.

#### ◆ Scripts

Script files contain OPLScript, a script language for OPL. A script handles different models with their data. The model and data file are associated in the script itself. For example, `mulprod.osc` contains the following declaration:

```
Model produce( "mulprod.mod", "mulprod.dat" ) editMode;
```

The following naming conventions are used to indicate these different files:

File Extension	Description
.mod	Used for files containing models.
.opl	Used for compiled model files.
.dat	Used for files containing data instances.
.prj	Used for project files.
.osc	Used for scripts written in OPLScript.

The examples in the tutorial chapters of this manual illustrate the use of model, data, project and script files with ILOG OPL Studio. You will see how to create project files, associate model and data files with the project, and then find the solution to the problem using the project file. The use of scripts is illustrated in Chapter 11, *Using OPLScript*.

## Opening an Existing File

The model and data files used in the examples in this manual are distributed with the product. In this way you will not have to create these files from scratch, but just open them once ILOG OPL Studio is launched.

To open an existing file, select Open>Model (or Data, Script, Project, C++ File) from the File menu. ILOG OPL Studio then displays a standard Open File dialog box for you to select the file you want to open.

Select from the directory:

ILOG\OPLSt37\examples\opl

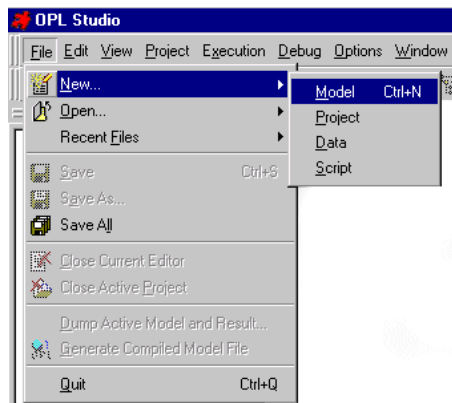
If you have recently used the file, you can alternatively select it from the Recent Files submenu.

## Creating a New File

When you are using ILOG OPL Studio to solve a problem of your own, you will first have to define a working document in ILOG OPL Studio. You can do this either by opening existing model and data files (that you created with a text editor of your choice) or by creating the file from scratch using the editing capabilities of ILOG OPL Studio.

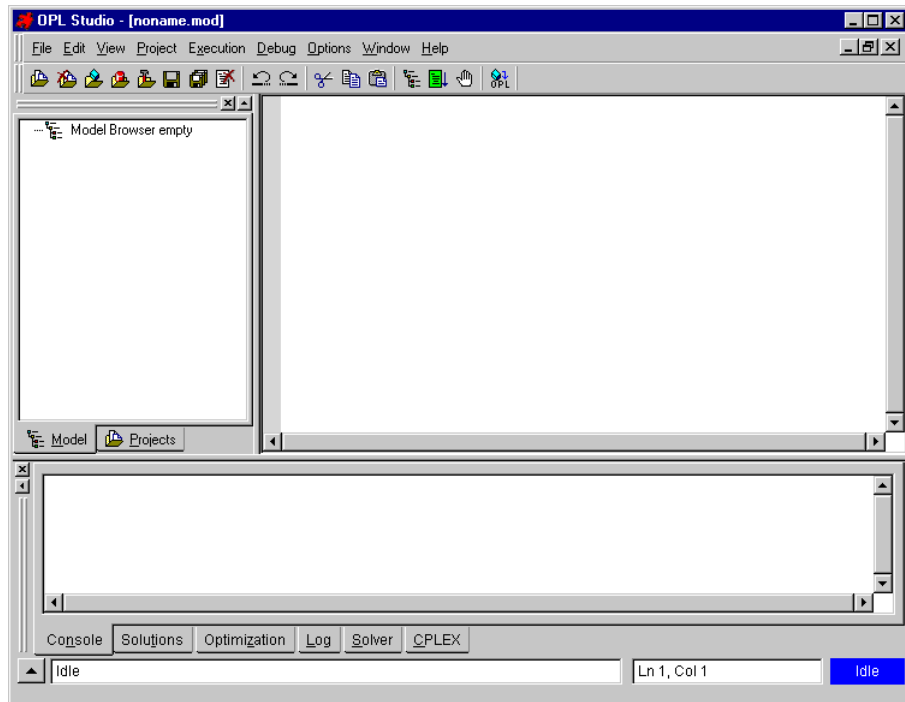
The following is a brief explanation of how to create a model file in ILOG OPL Studio.

From the File menu, select New>Model.



**Figure 1.7** Selecting a New Model

In the editing area of the Main window, ILOG OPL Studio opens an empty working document called `noname.mod`. The `.mod` extension indicates a model document.



**Figure 1.8** *New Model in Main Window*

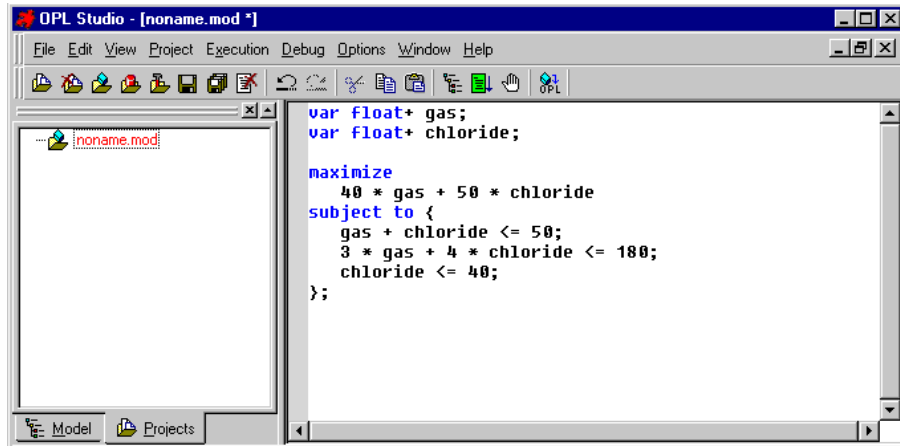
ILOG OPL Studio is now in editing mode and you can enter OPL statements for your problem. For example, if you want to create the model file for the simple production planning problem of the Volsay company that is presented at the beginning of Chapter 2 of the *ILOG OPL Studio: Language Manual*, the OPL statement for this problem is as follows:

```
var float+ gas;
var float+ chloride;

maximize
    40 * gas + 50 * chloride
subject to {
    gas + chloride <= 50;
    3 * gas + 4 * chloride <= 180;
    chloride <= 40;
};
```

**Code Sample 1.1** *Example of an OPL Statement*

When you finish, the editor looks something like this:



**Figure 1.9** Simple OPL Statement in the noname.mod file

You can now save your model under another name. From the File menu, select the Save As option. ILOG OPL Studio displays a standard Save As dialog box for you to supply the new file name.

When you save your file, the new name appears in the title bar of the Main window. You can now perform other tasks as required for this new working document.

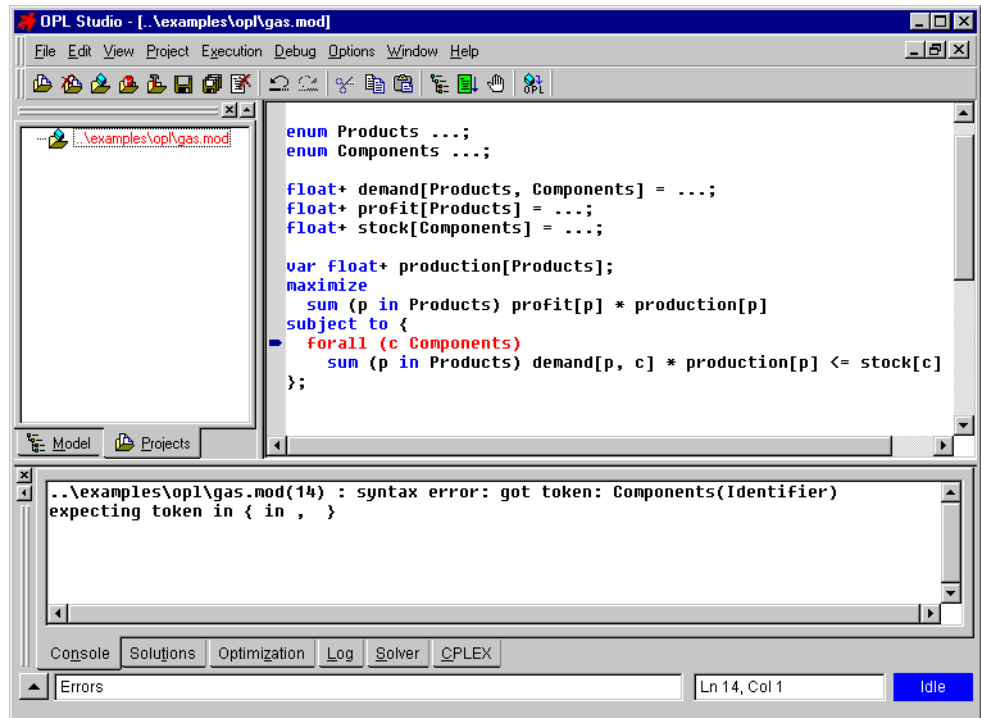
---

### Executing a Project or Model

Once a project or model is opened in ILOG OPL Studio, you can execute it by simply clicking the Run button  in the tool bar of the Main window.

## Checking for Syntactic or Semantic Errors

Before ILOG OPL Studio begins executing a model or project, it compiles the OPL statement into an internal representation that is better suited for execution. If the model contains syntactic or semantic errors, ILOG OPL Studio reports them immediately and does not continue with the execution. All errors appear in the Console notebook page of the output area in the lower half of the Main window. In addition, the line containing the error is highlighted in the current model file and a blue arrow appears in the margin. Figure 1.10 shows ILOG OPL Studio as it reports a syntactic error.



**Figure 1.10** Error Message Displayed in Console Notebook Page

An error message has the following format when displayed in the Console notebook page:

- ◆ The first element indicates where the error was detected. In Figure 1.10, the file and the line in the file are displayed:

```
..\examples\opl\gas.mod(14)
```

- ◆ The second element indicates the type of error. In our example:

```
syntax error
```

- ◆ The third element indicates the token that was encountered.

```
got token: Components(Identifier)
```

- ◆ The fourth element shows a list of tokens that could have been valid at that point. In our example:

```
expecting token in {in, }
```

An OPL statement must be correct before ILOG OPL Studio will execute it. You must correct any errors before running the model again.

**Note:** If there are multiple errors, double-click on an error message to scroll the editor to the corresponding line.

---

## Specifying Processing Directives

The following buttons in the tool bar of the Main window can be used to indicate how to proceed in the processing of a model or project:



- **Next**

Goes to the next solution of the model or project.



- **Continue Run**

Forces ILOG OPL Studio to produce all the remaining solutions without further intervention.



- **Abort**

Stops the current computation during execution of the model, project or script. After pressing the Abort button, the traces of any solutions found up to that point are kept in the Output notebook.

**Note:** The abort process can sometimes take a little time.

---

## Terminating ILOG OPL Studio

To terminate an ILOG OPL Studio session, select Quit from the File menu of the Main window.



## The Text Editor

The OPL Studio text editor has the following features:

- ◆ MDI approach

The MDI (Multi-Document Interface) enables you to edit more than one file at the same time.

- ◆ Syntax coloring

The syntax in each type of file that you can load (model, script, data or C++ file) is colored differently, according to its type.

- ◆ Multiple levels of Undo and Redo

You can undo and redo your modifications without any limit.

- ◆ Automatic indentation



{ } blocks are automatically indented.

- ◆ Brace matching

When typing ], } or ), the matching open brace is highlighted for 800 ms. In data files < and > are also matched.

- ◆ Margin symbols

The editor has a left margin that can contain margin symbols, such as:

- the yellow arrow that indicates the current line 
- the blue arrow that indicates an error 

- ◆ Reload prompt





If you modify a file with an external editor, you are prompted to reload the file as soon as the OPL Studio editor regains focus.

---


### Switching Between Editor Windows

The editor respects the Multi-Document Interface (MDI) approach, which allows you to edit more than one file at a time and thus have several document windows open simultaneously, as shown in Figure 1.2, *Main Window*.

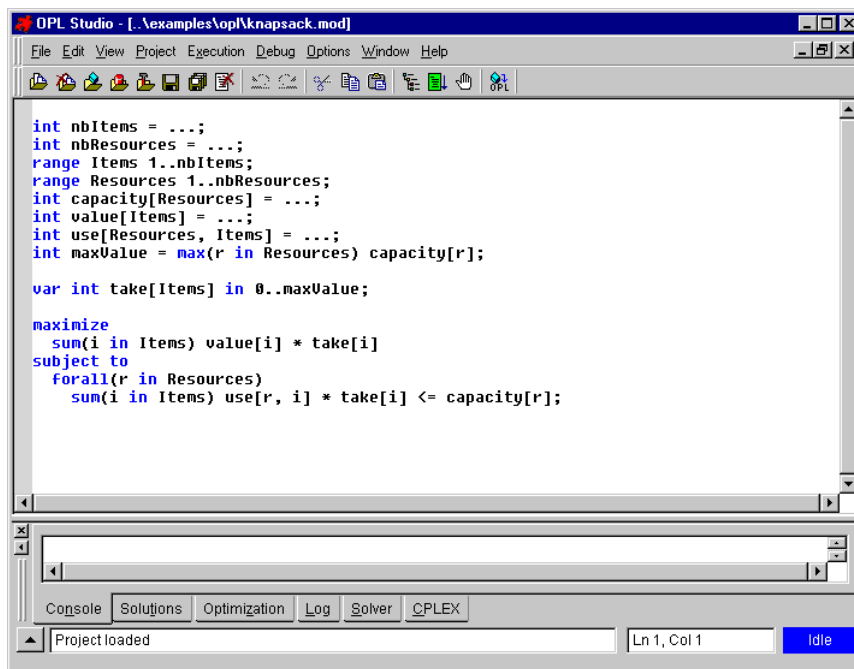
The Window menu lists the opened files, and allows you to arrange them in a cascade or as tiles. An icon in the upper-left corner of each window reminds you of the file type:

- model file 
- data file 
- script file 
- C++ file 

## Resizing an Editor Window

An editor window may have three states: minimized, normal, or maximized. By default, it is maximized. The buttons on the right-hand side of the menu bar  enable you to minimize, restore to normal size, or close the editor.

By hiding the docked windows you can use almost all the screen for editing, as shown in Figure 1.11.



**Figure 1.11** Text Editor after Hiding the Output and Model Windows

## Editor Quick Reference

In addition to the default Visual Mode (see *Setting Editor Options* on page 166), an Emacs mode, that emulates Emacs key bindings, is available.

### Note:

- ◆ Ctrl = Control
- ◆ Ctrl + x = Control and x keys simultaneously

**Table 1.2** *Editor Functions*

Action	Visual Mode	Emacs Mode
Save	Ctrl + s	Ctrl + x, Ctrl + s, or ESC x save-buffer
Save All		Ctrl + x s, or ESC x save-some-buffers
Save As		Ctrl + x, Ctrl + w, or ESC x write-file
Undo	Ctrl + z	Ctrl + x u, or Ctrl + _, or ESC x undo
Redo	Ctrl + y	Ctrl + x r
Next character (right)	→	Ctrl + f, or →, or ESC x forward-char
Previous character (left)	←	Ctrl + b, or ←, or ESC x backward-char
Next line (down)	↓	Ctrl + n, or ↓, or ESC x next-line
Previous line (up)	↑	Ctrl + p, or ↑, or ESC x previous-line
Next screen (down)	Page Down	Ctrl + v, or PageDown, or ESC x scroll-down
Previous screen (up)	Page Up	ESC v, or PageUp, or ESC x scroll-up
Beginning of file	Ctrl + Home	ESC <, or Home, or beginning-of-buffer

**Table 1.2** *Editor Functions (Continued)*

Action	Visual Mode	Emacs Mode
End of file	Ctrl + End	ESC >, or End, or ESC x end-of-buffer
Next word (right)	Ctrl + →	Ctrl + →, or ESC f, or ESC x forward-word
Previous word (left)	Ctrl + ←	Ctrl + ←, or ESC b, or ESC x backward-word
Beginning of line	Home	Ctrl + a, or ESC x beginning-of-line
End of line	End	Ctrl + e, or ESC x end-of-line
Delete character after cursor	Delete	Ctrl + d, or Delete, or ESC x delete-char
Delete character before cursor	Backspace	Backspace, or ESC x delete-backward-char
Cumulative cut from cursor to end of line	Ctrl + k	Ctrl + k, or ESC x kill-line
Cut	Ctrl + x or Shift + Delete on selection	Ctrl + w on selection, or ESC x kill-region
Copy	Ctrl + c or Ctrl + Insert on selection	ESC w or Ctrl + Insert on selection, or ESC x kill-ring-save
Paste	Ctrl + v, or Shift + Insert	Ctrl + y, or Shift + Insert, or right click, or ESC x yank
Scroll down one line	Ctrl + ↓	Ctrl + ↓
Scroll up one line	Ctrl + ↑	Ctrl + ↑
Recenter on current line	Ctrl + l	Ctrl + l, or ESC x recenter
Mark the beginning of a selection	Click left mouse button and drag cursor, or use arrow keys	Ctrl + SPACE, or ESC x set-mark-command

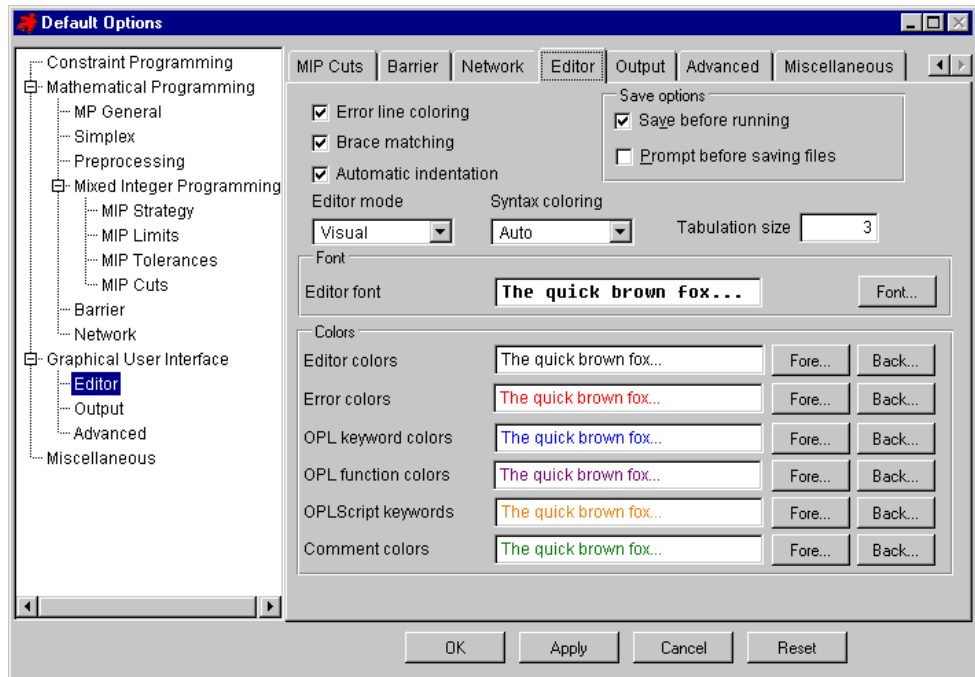
**Table 1.2** Editor Functions (Continued)

Action	Visual Mode	Emacs Mode
Mark the end of a selection or extend a selection	Release left mouse button, or press Shift + left button, or Shift + arrow keys, or Shift + Page keys, or Ctrl + Shift + arrow keys, or drag mouse	Change the caret location after marking the beginning of the selection
Stop command in progress	Release selection	Ctrl + g, or ESC x keyboard-quit
Select line	Click on the margin or triple click	Click on the margin or triple click
Select word	Double click on word	Double click on word
Select all	Ctrl + a or quadruple click	ESC + h, or quadruple click, or ESC x mark-whole-buffer
Switch editor buffers	Ctrl + Tab	Ctrl + Tab, or ESC x switch-to-buffer
Indent a region	Tab on selected lines	Ctrl + x / on selected lines, or ESC x indent-region
Unindent a region	Shift + Tab on selected lines	Ctrl + x m on selected lines, or ESC x back-to-indentation
Indent at location	Tab without any selection	Tab without any selection
Unindent at location	Shift + Tab without selection	Shift + Tab without selection
Comment lines		ESC x comment-region
Uncomment lines		ESC x kill-comment
Find	Ctrl + f	Ctrl + s, or ESC x search-forward
Find Next	F3	F3
Find Previous	Shift + F3	Ctrl + r
Replace	Ctrl + h	ESC % or ESC x query-replace
Complete Word	Ctrl + t	ESC / or ESC x dabbrev-expand
Go to a specific line	Ctrl + g	ESC g or ESC x goto-line

## Customizing the Editor

To customize the editor, select Options>Customize Default Options from the tool bar. Then go to the Editor page in the Default Options dialog box:

- either by clicking on Editor in the left panel
- or by using the next and previous arrows ◀ ▶ in the top right corner.



- ◆ You can change the character font. Note that although proportional fonts are allowed, we recommend that you use a fixed font.
- ◆ You can change the background and foreground colors for:
  - ordinary text (default: black and white)
  - errors (default: red and white)
  - OPL and C++ keywords (default: blue and white)
  - OPL and OPLScript functions (default: violet and white)
  - OPLScript keywords and C++ preprocessor macros (default: orange and white)
  - comments (default: green and white)

- ◆ You can switch off the error line coloring, the syntax coloring, and the brace matching.  
By default, the syntax coloring is automatically switched off if the file size is more than 65 KB. However, you can force the syntax coloring to be on or off.
- ◆ You can switch off the automatic indentation and change the tabulation size, but these two switches block the undo/redo mechanism for previous commands.  
Tabulations and indentations are handled as blanks.
- ◆ You can change the save preferences:
  - Save before running (checked by default)  
If checked, all edited files and projects are saved before execution.
  - Prompt before saving files (unchecked by default)  
If checked, a dialog box is displayed for you to confirm that you want to save a file.

## The Online Help

The online help provides a quick reference to OPL and OPLScript instructions, functions and methods.

### Windows Platforms



The online help available on Windows platforms is based on Microsoft HTML Help. To access the help, you must use Microsoft Internet Explorer 4.0 or above, and at least version 1.22 of the ActiveX control file HHCTRL.OCX must be installed.

If you are unsure of what components are installed on your system, you can upgrade it with the HTML Help update file (hhupd.exe) which contains browser and software compatibility updates. It is available for all languages at the following URL:

<http://msdn.microsoft.com/library/tools/htmlhelp/wkshp/hhupd.exe>



On windows platforms, the online help provides the following features:

- ◆ Integration of the entire online documentation set
  - *ILOG OPL Studio: User's Manual*
  - *ILOG OPL Studio: Language Manual*
  - *ILOG OPL Studio: Component Libraries User's Manual*
  - *ILOG OPL Studio: Component Libraries Reference Manual*
  - *ILOG OPL Studio: Release Notes.*
- ◆ A link to the OPL Studio home page on the ILOG web site (when connected to the Internet)
- ◆ A link to the model library on the ILOG web site (when connected to the Internet). You can download a model and use it as a basis for creating your own model.

To access the help from OPL Studio:

- ◆ either use the Help menu
- ◆ or press the F1 key on a selected keyword in the text editor.

To navigate in the help, use the contents list, the index, or the search function.

- ◆ Click on an item in the contents list to display the corresponding page.
- ◆ In the index field, type the beginning of a word, then double-click on the listed item you want to display.
- ◆ In the search field, type the word you want to find, then click on "List Topics". The search function lists all the pages that reference the word you entered (OPL keyword or natural language). When you double-click an item in the list, the page is displayed with all the referenced words highlighted.

---

## UNIX Platforms

**Prerequisite:** you need Netscape Communicator 4.0.

After clicking F1, if a Netscape window is available, the appropriate help page will be opened in the existing Netscape window. If not, you will be told to launch Netscape.

### **Important:**

- ◆ *If Netscape is launched from a different machine sharing the same display, ensure that the disk mount allows Netscape to find the directory in which OPL Studio is installed.*
- ◆ *It is recommended to launch Netscape after OPL Studio, as Netscape uses a large number of colors, leaving very few for OPL Studio.*



## ***Tutorial: Working with Projects***

This chapter provides an introduction to ILOG OPL Studio through the use of a simple production planning example.

In this example, you will learn how to:

- ◆ create a project and associate a model file and a data file with the project
- ◆ execute the project
- ◆ display and examine the results of the solution
- ◆ use the model browser
- ◆ close the project file with its associated model and data file
- ◆ work with several projects.

For this part of the tutorial, you will need the `product.mod` and `product.dat` files from your release distribution. If you used the default directories at installation time, you will find these files at the following location:

- ◆ For UNIX systems  
`<installation-directory>/OPLSt37/examples/opl`
- ◆ For Windows XP, Windows 2000, Windows NT 4, and Windows 98  
`c:\ILOG\OPLSt37\examples\opl`

## The Production Planning Example

The first example centers around the production planning model that appears in Chapter 2 of the *ILOG OPL Studio: Language Manual*. The problem is described as follows.

To meet the demands of its customers, a company manufactures its products in its own factories (*inside* production) or buys the products from other companies (*outside* production). The inside production is subject to resource constraints: each product consumes a certain amount of each resource. In contrast, the outside production is theoretically unlimited. The problem is to determine how much of each product should be produced inside the company and how much outside, while minimizing the overall production cost, meeting the demand, and satisfying the resource constraints.

Code Sample 2.1 shows the OPL model for this example. This model is found in the distributed `product.mod` file.

```
enum Products ...;
enum Resources ...;

struct ProductData {
    float+ demand;
    float+ insideCost;
    float+ outsideCost;
    float+ consumption[Resources];
};

ProductData product[Products] = ...;
float+ capacity[Resources] = ...;

var float+ inside[Products];
var float+ outside[Products];

minimize
    sum(product[p].insideCost*inside[p] +
        product[p].outsideCost*outside[p])
subject to {
    forall(r in Resources)
        sum(p in Products) product[p].consumption[r] * inside[p] <= capacity[r];
    forall(p in Products)
        inside[p] + outside[p] >= product[p].demand;
};
```

**Code Sample 2.1** OPL Model for the Production Planning Example (`product.mod`)

Code Sample 2.2 shows the data initialization for the problem. This can be found in the `product.dat` file of the product distribution.

```
Products = {kluski capellini fettucine};
Resources = {flour eggs};
product =
  #[
    kluski : < 100, 0.6, 0.8, [0.5, 0.2] >
    capellini : < 200, 0.8, 0.9, [0.4, 0.4] >
    fettucine : < 300, 0.3, 0.4, [0.3, 0.6] >
  ]#;
capacity = [20, 40];
```

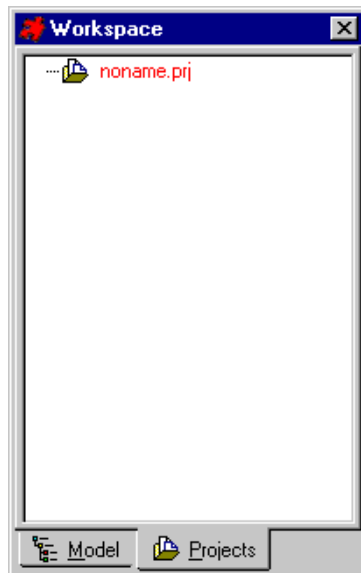
**Code Sample 2.2** *OPL Data for the Production Planning Example* (`product.dat`)

Your first task in this tutorial will be to create a project file that associates these two files, `product.mod` and `product.dat`. You will then see how to execute the project, look at the results of the solution, and close the project.


## Creating a Project

After launching ILOG OPL Studio as described in *Launching ILOG OPL Studio* on page 18, create the project file that will contain the `product.mod` and `product.dat` files by selecting **New>Project** from the File menu.

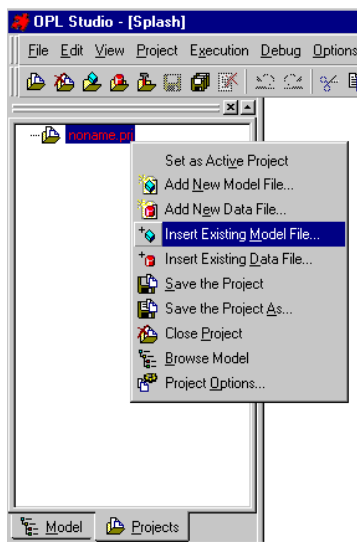
ILOG OPL Studio displays the project tree in the work space, to which you can add all files related to the project. Notice that it creates a project called `noname.prj` at the top of the tree structure. Project files are displayed in the Projects page and use the `.prj` extension.




**Figure 2.1** Project Window (Floating State)

To close the active project, use either the command **File>Close Active Project**, or the button  in the main tool bar.

## Inserting an Existing Model File into a Project



**Figure 2.2** Insert a Model File into a Project

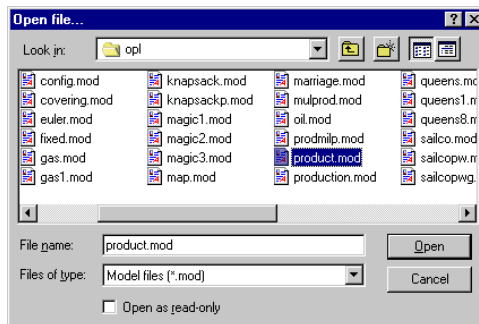
To insert an existing model into the new project, position the cursor on the project name (`noname.prj`) and right-click on the mouse. Select  Insert Existing Model File from the popup menu, as shown in Figure 2.2.

The ILOG OPL Studio distribution structure contains five directories:

```
bin
doc
examples
include
lib
```

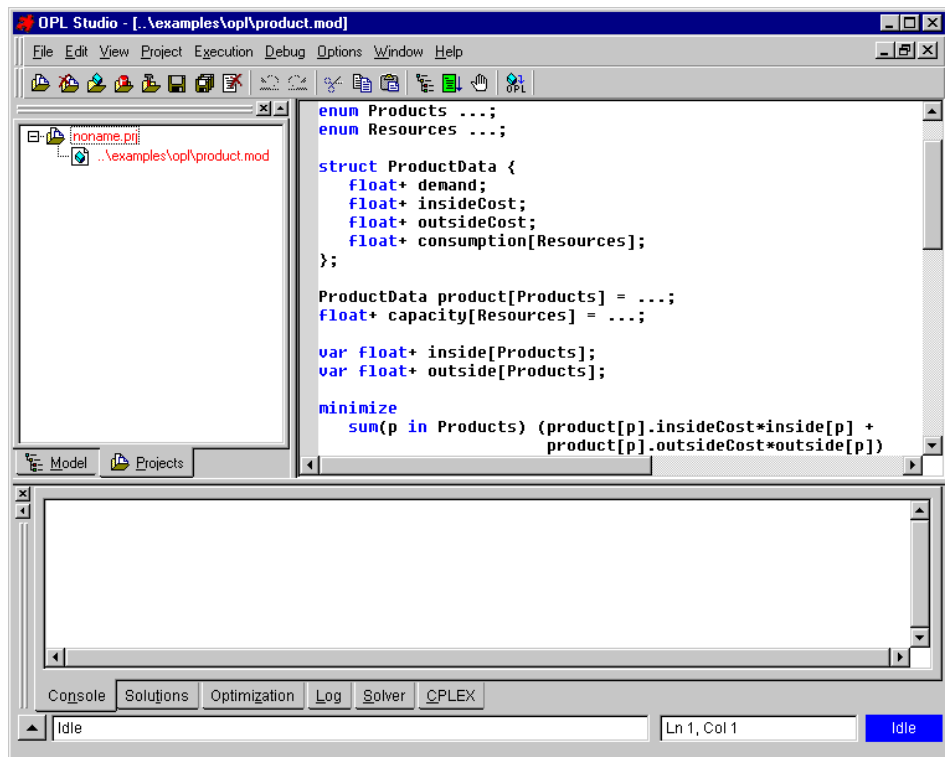
You will find the list of available model files in:

```
<installation-directory>\examples\opl.
```



Select the `product.mod` file from the list and click Open.


Notice that ILOG OPL Studio adds the `product.mod` file to the project tree and opens the file in the work space.



**Figure 2.3** Project Tree and Editing Area After Inserting a Model File



## Inserting an Existing Data File into a Project

Next, you are going to insert the `product.dat` file into the project. Position the cursor on the project name (`noname.prj`) and right-click on the mouse. Select  Insert Existing Data File from the menu, as shown in Figure 2.4.

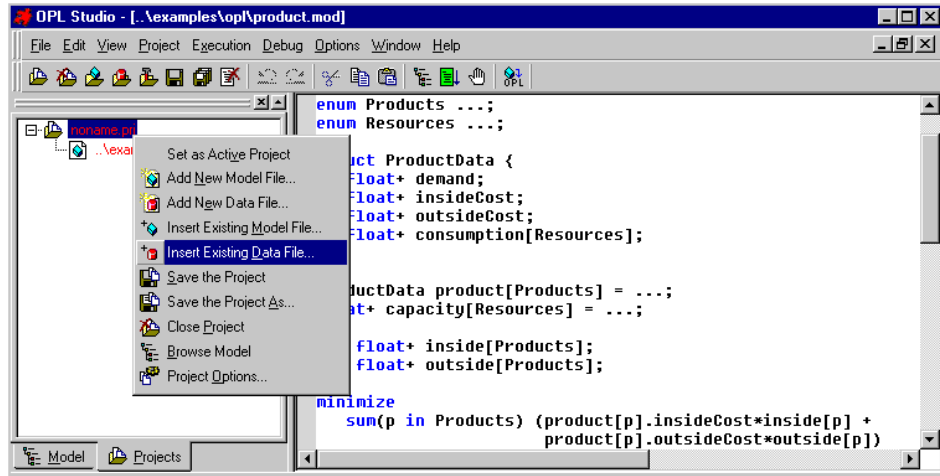


Figure 2.4 Inserting a Data File into a Project

ILOG OPL Studio again displays the Open File dialog box, this time filtering the `.dat` files.

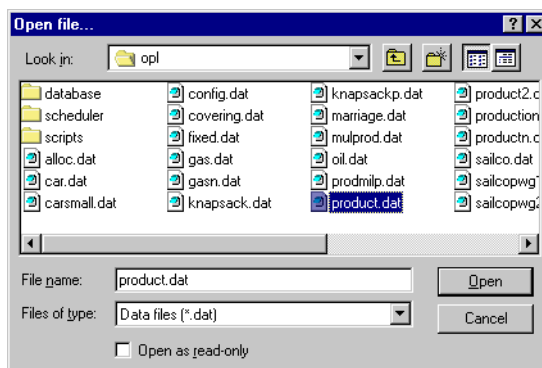
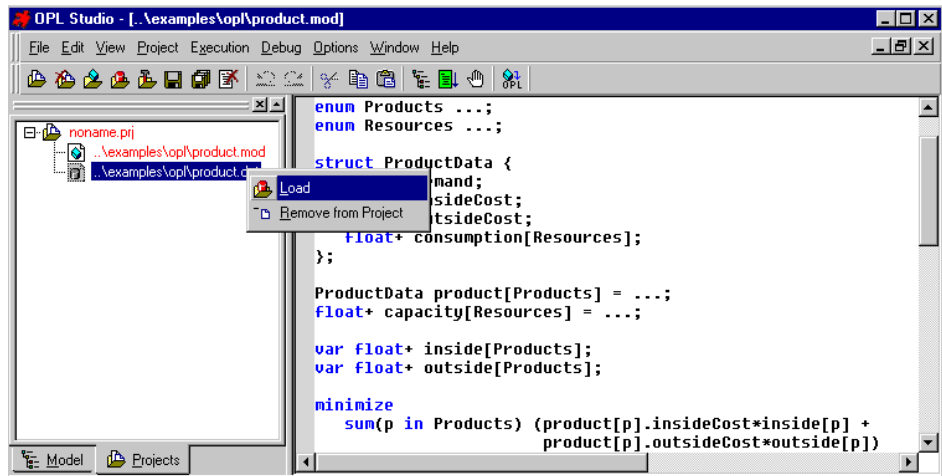


Figure 2.5 Open Data File Dialog Box

Select the `product.dat` file from the list and click Open.

ILOG OPL Studio adds the `product.dat` file to the tree structure. To load the data into the editor, right-click on the gray data icon then click on the Load button that appears. This opens the data file in the editing area.



**Figure 2.6** Loading Data into a Project

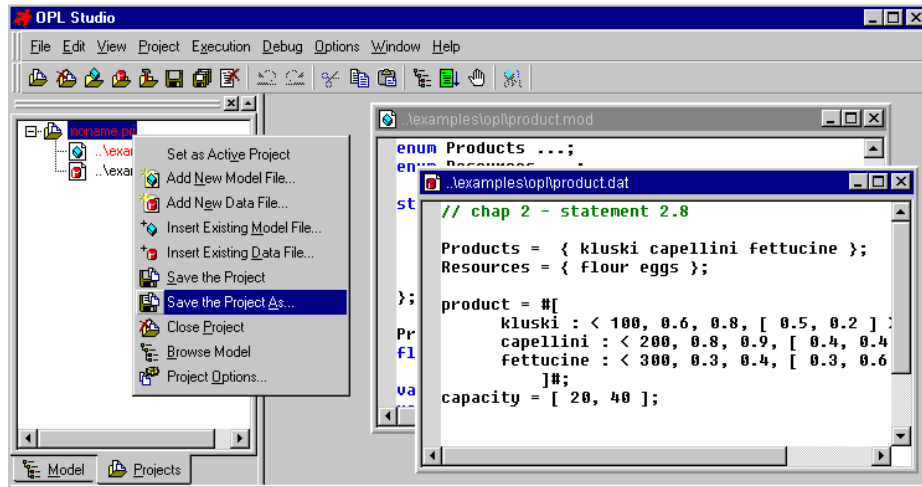
You can create a project without loading the data file into the editor. This feature is useful if you have a large data file that takes a long time to load.

### Adding New Files to a Project


If you create a new model file, or data file, and want to add it to a project, click on the project name and select, as appropriate:

- ◆ Add New Model File
- ◆ Add New Data File.

## Saving the Project

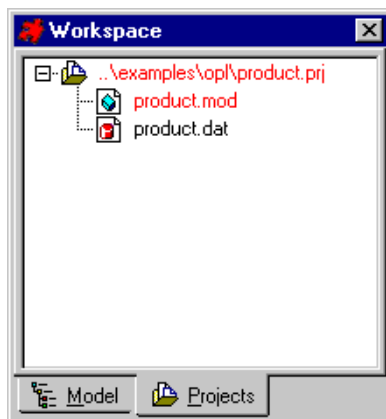


**Figure 2.7** Saving a Project

To save your project file, position the cursor on the project name (`noname.prj`) and right-click on the mouse. Select the menu item  Save the Project As (see Figure 2.7).

ILOG OPL Studio displays a standard Save As dialog box. Enter `product.prj` for the file name and click Save. (As `product.prj` already exists in your distribution, you will have to overwrite it.)


Notice that `noname.prj` at the top of the tree has now changed to `product.prj`.



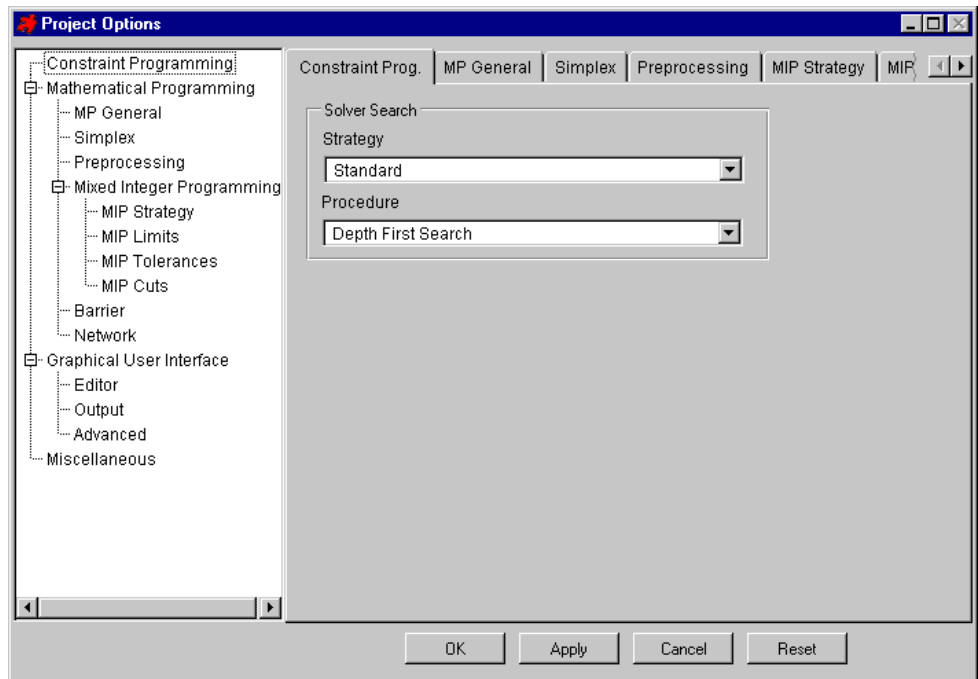
**Figure 2.8** Project Tree for `product.prj` Example

## Setting Project Options

ILOG OPL Studio also allows you to set certain options for your project.

- ◆ Right-click on the project name in the project tree, then select  Project Options
- ◆ or select Options>Customize Active Project Options from the menu bar.

ILOG OPL Studio displays the Project Options notebook.




**Figure 2.9** Project Options Notebook

To navigate through the notebook, you can either click on elements in the tree structure or click on a tab. Within the notebook pages, you can set options for:

- ◆ constraint programming (for ILOG Solver and ILOG Scheduler)
- ◆ linear optimization (for ILOG CPLEX)
- ◆ optimization using simplex (for ILOG CPLEX)
- ◆ mixed integer programming (for ILOG CPLEX)
- ◆ the barrier algorithm (for ILOG CPLEX)
- ◆ the network simplex algorithm (for ILOG CPLEX)
- ◆ the text editor
- ◆ labels and output
- ◆ paths for OPL and OPLScript files, and for compiled model generation.

To set the project options, click Apply then OK in the notebook.


To save these options, right-click on the project name in the project tree, then select  Save the Project.

The new options are stored in the current `.prj` file and restored automatically when the project is re-opened. They do not affect files that are not associated with the current project.

To set default options for all files, see Chapter 8, *Customizing ILOG OPL Studio*.

For the present production planning example, you are going to use the delivered defaults, so just click Cancel.

## Executing the Project

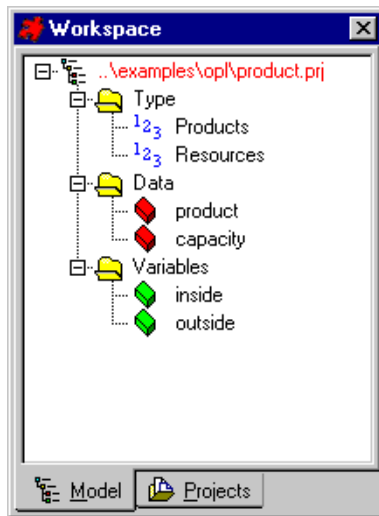
As the project file `product.prj` is now open, you can execute the project and find the solution. Click the Run button  in the tool bar of the Main window.

You will notice that the execution tool bar is displayed.

**Note:** ILOG OPL Studio always gives precedence to the active project. If you have other models, scripts or projects open in the Main window, the model of the active project always gets executed. To switch to another model, project or script, right-click on it in the project tree and select the menu item *Set As Active*.

When you execute a project, you trigger a chain of events.

ILOG OPL Studio first analyzes the model and produces summary information. Click on the tab *Model* to display the model browser in the Workspace window. This browser contains information about the data structures defined in the model. The model browser is described in *Using the Model Browser* on page 67.

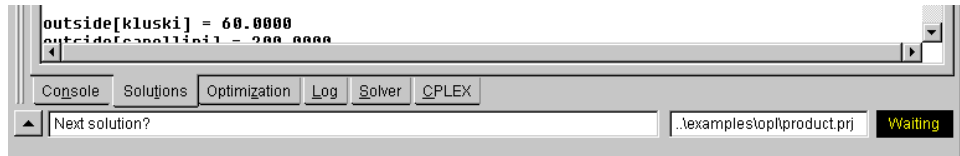


**Figure 2.10** Model Browser for `product.prj` Example

ILOG OPL Studio also checks for syntactic or semantic errors at this time. If it finds errors, it stops the execution and the errors must be corrected. See *Checking for Syntactic or Semantic Errors* on page 39.

ILOG OPL Studio then executes the model. The message “OPL Studio is running” appears in the status bar, the name `product.prj` appears in the Path Name area, and the color patch turns to green to indicate the program is running.

When ILOG OPL Studio finds a solution, it enters a waiting state and expects further instruction from you. The message “Next solution?” appears in the status bar, and the color patch turns to yellow and blinks to indicate the waiting state.



**Figure 2.11** “Next solution?” Message

You can use this break in the execution to examine the solution in more detail.

## Examining a Solution to the Model

ILOG OPL Studio provides two facilities for examining the details of your model and the solutions:

- ◆ the output notebook in the lower half of the Main window
- ◆ the model browser in the work space.

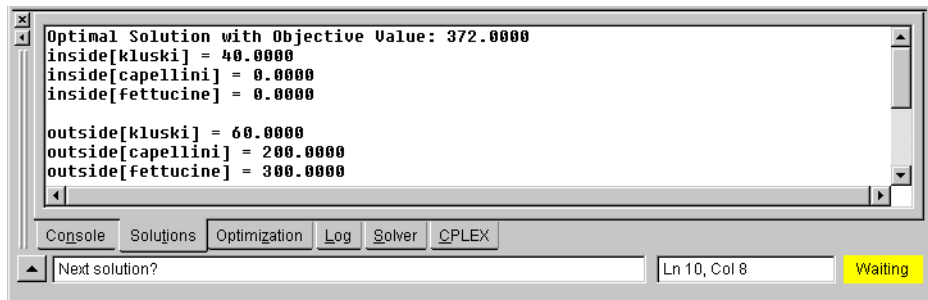
### Using the Output Area

When a solution to the model is found, ILOG OPL Studio fills in several notebook pages in the output area. These pages show:

- ◆ the solution that ILOG OPL Studio has found
- ◆ optimization information (if it exists)
- ◆ log information
- ◆ solving statistics.

### Solutions Notebook Page

The Solutions notebook page is used to display the current solution to the model.



**Figure 2.12** Solutions Notebook Page for `product.prj` Example

By default, all variables are displayed. However, you can specify which variables are displayed and in what format. See Chapter 9 of the *ILOG OPL Studio: Language Manual* for more information.

### Optimization Notebook Page

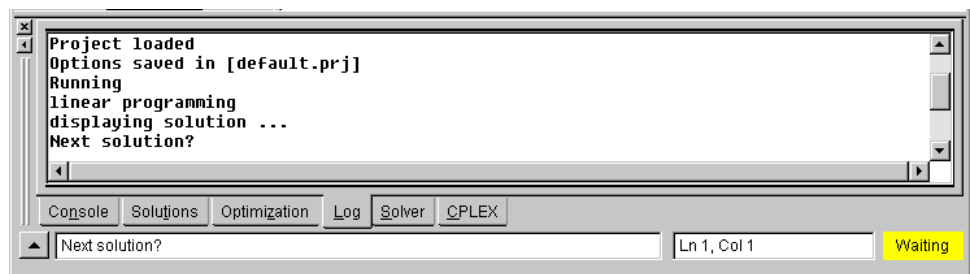
When you click the Optimization tab, you will see that the notebook page is empty. The Optimization notebook page contains information:

- ◆ only when there is an optimization statement in the model (that is, a minimize or maximize statement)
- ◆ and only when optimization is obtained by improving on successive solutions.

When these two conditions are met, all the solutions found (and their costs) are displayed. If the model contains only a solve statement, the first solution is supplied in the Solutions page and the Optimization page is empty.

### Log Notebook Page

Click either the Log tab, or the arrow at the left of the status bar, to view log details.



**Figure 2.13** Log Notebook Page for `product.prj` Example

The information displayed in the log notebook page is of the following type:

- ◆ status changes



- ◆ file loadings
- ◆ results of analyzing a model.

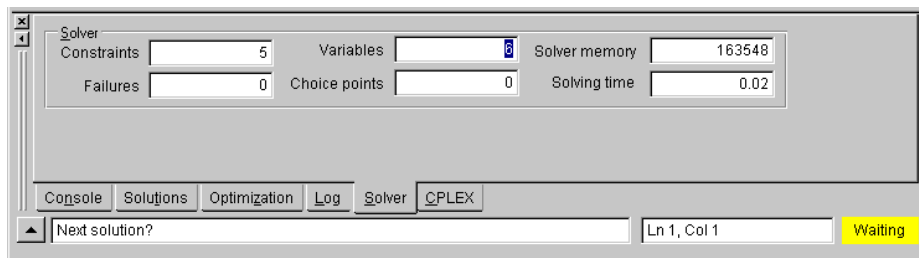
You can consult this log information to find out which kind of algorithm OPL Studio has switched to when analyzing a model. In our example, the Log notebook page displays the "linear programming" message. OPL Studio has detected a linear problem so it uses ILOG CPLEX as the solving engine.

The type of algorithm used varies according to the model analyzed, and can be one of the following:

- ◆ linear programming (CPLEX)
- ◆ piecewise linear programming (CPLEX MIP)
- ◆ piecewise linear programming (Solver MIP)
- ◆ integer programming (CPLEX MIP)
- ◆ integer programming (Solver)
- ◆ integer programming (Solver MIP)
- ◆ mixed integer programming (CPLEX MIP)
- ◆ mixed integer programming (Solver MIP)
- ◆ Solver
- ◆ Solver + Scheduler
- ◆ Solver + Hybrid
- ◆ Solver + Hybrid + Scheduler

### Solver Notebook Page

Click the Solver tab to view statistics that ILOG OPL Studio gathered while solving the model. They correspond to ILOG Solver library statistics.



**Figure 2.14** Solver Notebook Page for product.prj Example

ILOG OPL Studio reports the following statistics in the Solver notebook page:

- ◆ **Constraints** - the number of constraints it received

- ◆ **Failures** - the number of failures encountered during the resolution
- ◆ **Variables** - the number of variables received
- ◆ **Choice points** - the number of choices needed to produce the solution
- ◆ **Solver memory** - the amount of memory used
- ◆ **Solving time**
  - computation time on UNIX platforms
  - elapsed time (computation plus graphics display) on PCs.

In our example the number of choice points is zero since ILOG Solver's constraint programming search is not used here (`product.prj` is a pure LP model).

### CPLEX Notebook Page

Primal Phase I Infeasibility <input type="text"/> Iterations <input type="text"/>		Dual Phase I Infeasibility <input type="text"/> Iterations <input type="text"/>		Barrier Primal objective <input type="text"/> Dual objective <input type="text"/> Iterations <input type="text"/>		MIP Nodes <input type="text"/> Nodes left <input type="text"/> Iterations <input type="text"/> Best <input type="text"/> Bound <input type="text"/> Cutoff <input type="text"/>	
Primal Phase II Objective <input type="text"/> Iterations <input type="text"/>		Dual Phase II Objective <input type="text" value="3.72e+002"/> Iterations <input type="text" value="7"/>		Constraints <input type="text" value="5"/> Variables <input type="text" value="6"/>			
Primal Crossover Push <input type="text"/> Exchange <input type="text"/>		Dual Crossover Push <input type="text"/> Exchange <input type="text"/>					

Dual phase II

Console Solutions Optimization Log Solver CPLEX


Next solution?  Ln 1, Col 1 Waiting

**Figure 2.15** CPLEX Notebook Page for `product.prj` Example

This page shows CPLEX statistics. Here, in dual phase II, the objective is 3.72e+002, and there are 7 iterations, 5 constraints and 6 variables.

## Using the Model Browser

To browse the active model in the work space, you can choose from one of the following methods:

- ◆ In the tool bar:
  - Click on the button  Rebuild Browser Information
  - Select Execution>Browse Active Model
- ◆ In the model browser, right click on the root item and select Browse Active Model
- ◆ In the project tree, right-click on the project name and select Browse Model.

The model browser provides another way for you to examine the solution to your model. ILOG OPL Studio displayed the model browser before it began executing the model and so it is visible during the execution process. The model browser summarizes information about the data structures defined in the model. It lists the types and constants that appear in the model, as well as any variables, activities, or resources. As you can see, the following have been defined for the model `product.mod`:

- ◆ Products and Resources in the Type folder
- ◆ `product` and `capacity` in the Data folder
- ◆ `inside` and `outside` in the Variables folder.

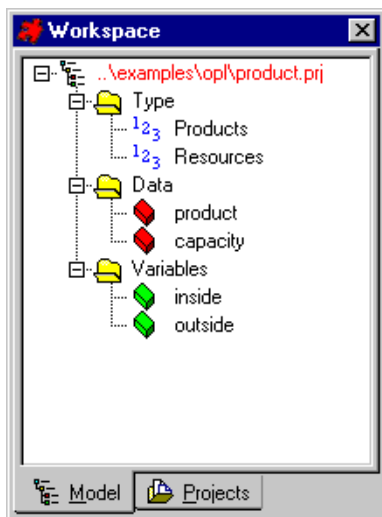


Figure 2.16 Model Browser for `product.mod` File

The model browser can be used in three different ways:

- ◆ To help you navigate through a .mod file displayed in the editing area
- ◆ To display additional views of the solution after running OPL Studio
- ◆ To check dynamic display options on variable items before running OPL Studio. These options are not available when OPL is in a waiting state, as it is now. Dynamic display is explained in *Using Dynamic Display with ILOG OPL Studio* on page 90.

### Navigating the .mod File

If you click on one of the entries in the project tree, ILOG OPL Studio selects the line in `product.mod` containing the first occurrence of that entry. For example, click on `capacity` and the first occurrence of `capacity` is highlighted.

### Displaying Additional Views

If you right-click on `capacity` while OPL Studio is in a waiting state, the Open View button appears, as shown in Figure 2.17. By clicking this button, you can display an additional view of the solution.

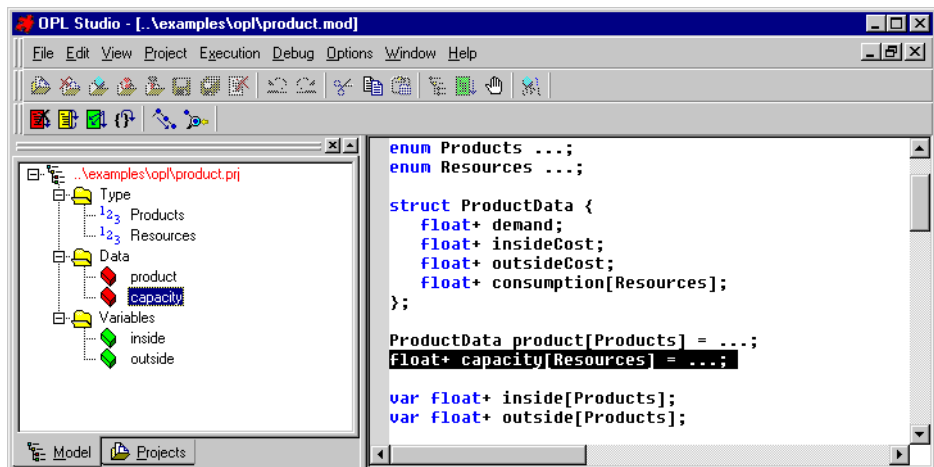
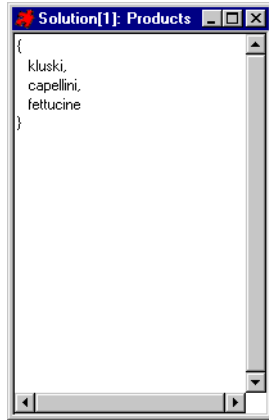


Figure 2.17 Open View Button

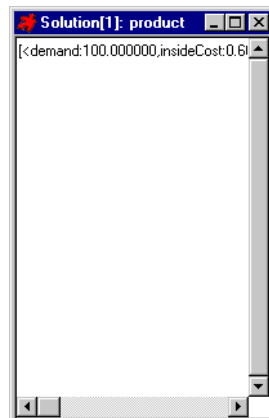
When you double-click an entry in the tree, ILOG OPL Studio displays an additional view for that entry and highlights the corresponding declaration in the text editor. The presentation of the object's content depends on the type of object displayed.

For example, if you double-click `Products` in the `Type` folder, ILOG OPL Studio displays the following view:

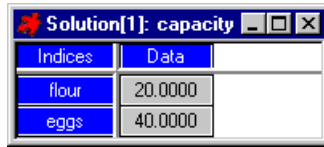


If you double-click `Resources` under `Type`, you will see a similar display with the defined resources.

If you double-click `product` in the `Data` folder, ILOG OPL Studio displays the following:



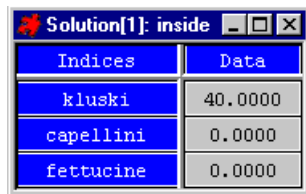
If you double-click `capacity` under `Data`, ILOG OPL Studio displays the following:



Indices	Data
flour	20.0000
eggs	40.0000

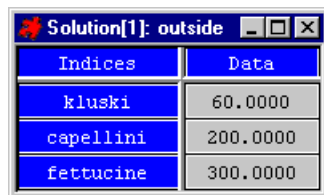
As you can see, the `Types` and `Data` panels are not very interesting since you can very readily see the same information in the `.mod` file or the `.dat` file. The panels displayed for `Variables` are more interesting since they represent the results of the problem. At times, you may want to display the `Data` panels and the `Variables` panels together so that the panels have the same appearance and can be compared easily.

If you double-click `inside` under `Variables`, ILOG OPL Studio displays:



Indices	Data
kluski	40.0000
capellini	0.0000
fettucine	0.0000

If you double-click `outside` under `Variables`, ILOG OPL Studio displays:



Indices	Data
kluski	60.0000
capellini	200.0000
fettucine	300.0000

In the other examples in this manual, you will see how ILOG OPL Studio uses other objects to display the results of a solution. But for now, let's continue with our production planning project.

## Continuing the Execution

All the while you were examining the model, ILOG OPL Studio was in its waiting state, expecting you to tell it what to do next. The following buttons in the execution tool bar of the Main window allow you to continue:

- **Abort**



Terminates the execution. After an Abort, the traces of any solutions found up to that point are kept in the Output notebook.

- **Next**



Goes to the next solution of the model.

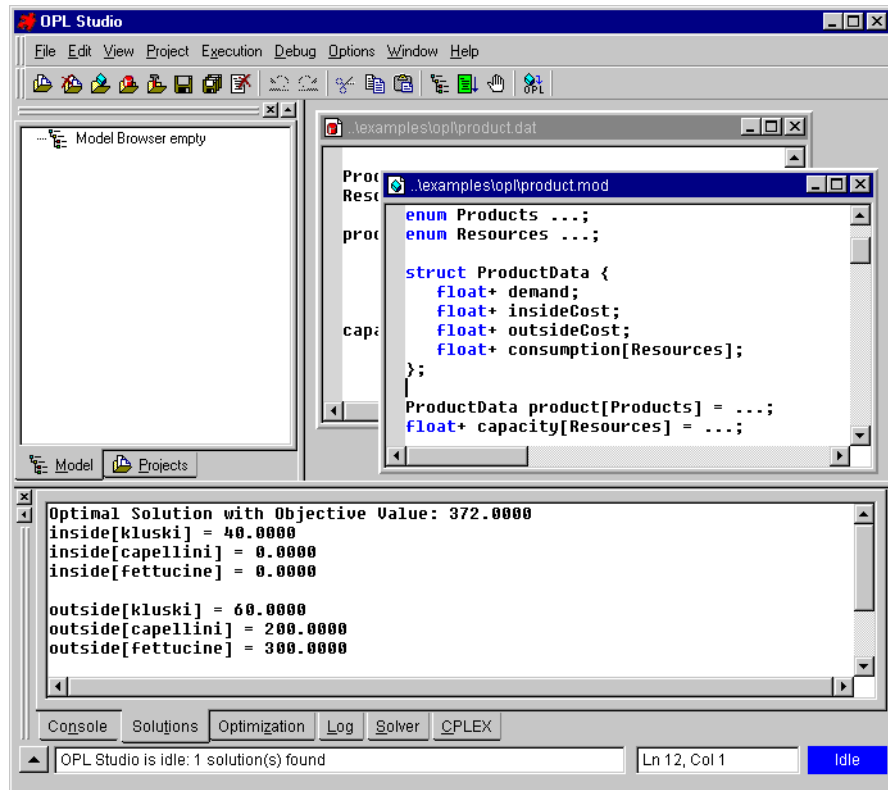
- **Continue Run**



Forces ILOG OPL Studio to produce all the remaining solutions without further intervention.

For this example, let's go to the next solution. Click the Next button in the tool bar.

ILOG OPL Studio continues its computation. Since there is only one solution to this model, it completes its execution and returns to its idle state. The status bar displays “OPL Studio is idle: 1 solution(s) found” and the color patch returns to blue.



**Figure 2.18** Main Window After Executing product.prj Example

Notice also that in the model browser, the tree structure has collapsed and no longer shows the data structures.



## Using an Alternative Data File

Here we examine an alternative way of expressing the data initialization of this problem. In the `product.dat` file, the expression was a simple initialization of records. OPL provides the possibility of a named initialization of records, as shown in Code Sample 2.3.


```
Products = {kluski capellini fettucine};
Resource = {flour eggs}
product =
#[
    kluski :
        #< demand:100
            insideCost:0.6
            outsideCost:0.8
            consumption: [0.5 0.2]
        >#
    capellini :
        #< demand:200
            insideCost:0.8
            outsideCost:0.9
            consumption: [0.4 0.4]
        >#
    fettucine :
        #< demand:300
            insideCost:0.3
            outsideCost:0.4
            consumption:[0.3 0.6]
        >#
    ]#;
capacity = [20, 40];
```

**Code Sample 2.3** OPL Named Data for Production Planning Example (`productn.dat`)

To use this alternative data file, you can open the `product.prj` file, remove the `product.dat` file (see *Removing a File from a Project* on page 75), and insert the file `productn.dat`. When you execute `product.prj` this time with the `productn.dat` file, you should see exactly the same results as when `product.dat` was associated with it.

## Closing a Project File

This completes the production planning example. Since you are finished with the `product.prj` file, you can close it, in one of three ways:

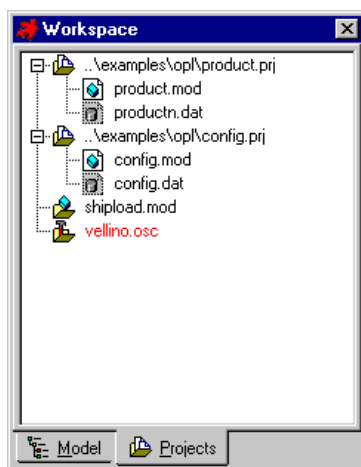
- ◆ Select **File>Close Active Project** from the menu bar, because `product.prj` is the active project
- ◆ Click on the **Close Active Project** button  in the tool bar, as `product.prj` is the active project
- ◆ Right click on `product.prj` in the project tree, then select **Close Project**.

You will notice that when you close the project, you also close all the files associated with it.

## Working with Several Projects

ILOG OPL Studio allows you to load more than one project at the same time. Of course, only one of the loaded projects is the active one. The active project name and its model name are displayed in red.

A project tree structure is displayed in the work space:



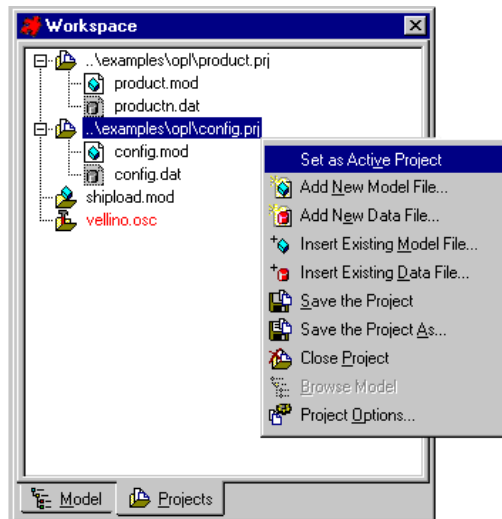
**Figure 2.19** Project Tree in the Work Space

As shown in Figure 2.19, stand-alone models and scripts are also displayed on the Projects page, although they are not true projects.

You can set a stand-alone model as the active model, or a script as the active script. Stand-alone models do not have their own settings, they take them from the default options.

## Setting the Active Project

By default, the last loaded project is the active one. If you want to change the active project, right-click on the new project name in the Projects viewer and select the menu item Set As Active Project, as shown in Figure 2.20.

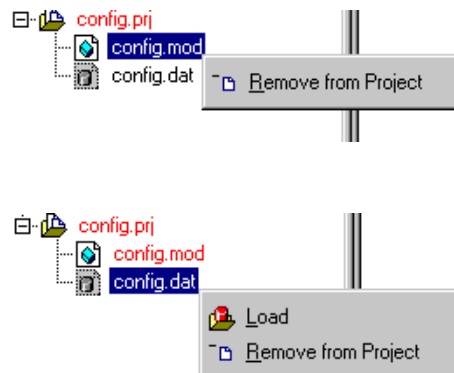


**Figure 2.20** Setting an Active Project

To change a project's options or close a non-active project, right click on the corresponding tree item.

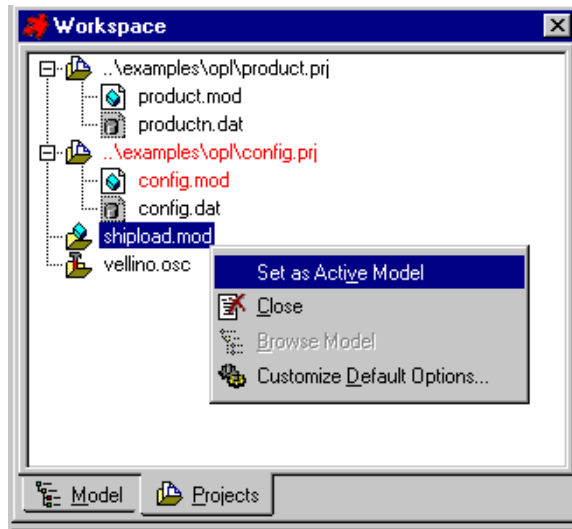
## Removing a File from a Project

To remove a model or data file from a project, right click on the corresponding file name and select Remove from Project.



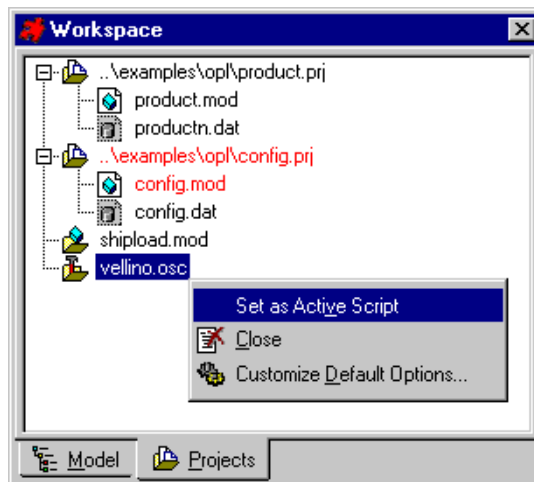
### Setting the Active Model

For a stand-alone model select Set As Active Model:



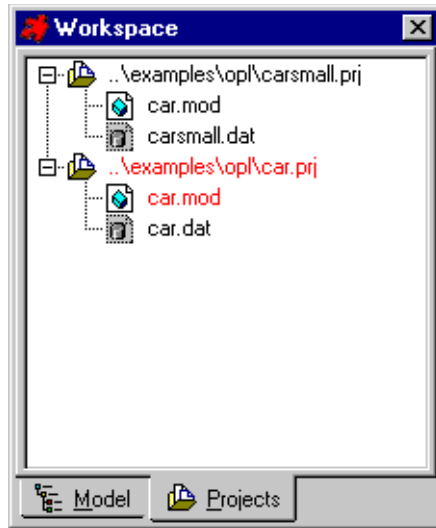
### Setting the Active Script

For a stand-alone script select Set As Active Script:



## Two References to the Same File

You can open two projects, standalone models or scripts that reference the same file.



In the figure above, the file `car.mod` is referenced by two different projects, `carsmall.prj` and `car.prj`.



## ***Tutorial: Predefined Dynamic Display***

In this chapter, you will see how ILOG OPL Studio handles a basic constraint model with constraint programming constructs.

Using the car sequencing example, you will:

- ◆ open a project file with its associated model and data files
- ◆ load the data into the editor
- ◆ execute the project
- ◆ display and examine the results of the model solution, noting how ILOG OPL Studio displays the variables of a solution
- ◆ complete the model execution by searching for all solutions to the problem
- ◆ generate summary information in the model browser and select dynamic display
- ◆ re-execute the project, using dynamic display to view solutions.

For this part of the tutorial, you will need these from your release distribution:

`car.mod`, `car.dat`, `car.prj`

If you used the default directories at installation time, you can find these files at the following location:

- ◆ For UNIX systems

`<installation-directory>/OPLSt37/examples/opl`

- ◆ For Windows XP, Windows 2000, Windows NT 4, and Windows 98

`c:\ILOG\OPLSt37\examples\opl`

---

## The Car Sequencing Example

This example centers around the car sequencing model that appears in Chapter 14 of the *ILOG OPL Studio: Language Manual*. The problem is as follows.

Cars in production are placed on an assembly line and move through various units that install various options, such as air conditioning or car radios. The assembly line is composed of slots and each car must be allocated to a single slot. The cars cannot be allocated arbitrarily because each production unit has a limited capacity and the options must be added to the cars as the assembly line passes in front of the unit. The capacity constraints for this problem are expressed using the form *l outof u*. They indicate that the unit can produce at most *l* cars with the option out of each sequence of *u* cars. The purpose is to find an assignment of cars to the slots of the assembly line that satisfies the capacity constraints.

Code Sample 3.1 shows the OPL model for this example. This file is `car.mod` of your release distribution.



```

int nbCars = ...;
int nbOptions = ...;
int nbSlots = ...;

range
    Cars 1..nbCars,
    Options 1..nbOptions,
    Slots 1..nbSlots;

int demand[Cars] = ...;
int option[Options,Cars] = ...;

struct Tcapacity {
    int l;
    int u;
};
Tcapacity capacity[Options] = ...;
int optionDemand[i in Options] = sum(j in Cars) demand[j] * option[i,j];

var
    Cars slot[Slots],
    int setup[Options,Slots] in 0..1;

solve {
    forall(c in Cars)
        sum(s in Slots) (slot[s] = c) = demand[c];

    forall(o in Options & s in [1..nbSlots - capacity[o].u + 1])
        sum(j in [s..s + capacity[o].u - 1]) setup[o,j] <= capacity[o].l;

    forall(o in Options & s in Slots)
        setup[o,s] = option[o,slot[s]];

    forall(o in Options & i in [1..optionsDemand[o]])
        sum(s in [1..nbSlots - i * capacity[o].u])
            setup[o,s] >= optionDemand[o] - i * capacity[o].l;
};

```

**Code Sample 3.1** OPL Model for the Car Sequencing Example (car.mod)

Code Sample 3.2 shows the data initialization for the problem. This code can be found in the `car.dat` file of the release distribution.

```
nbCars = 6;
nbOptions = 5;
nbSlots = 10;
demand = [1,1,2,2,2,2];
option = [
    [1, 0, 0, 0, 1, 1],
    [0, 0, 1, 1, 0, 1],
    [1, 0, 0, 0, 1, 0],
    [1, 1, 0, 1, 0, 0],
    [0, 0, 1, 0, 0, 0]
];
capacity = [
    <1,2>,
    <2,3>,
    <1,3>,
    <2,5>,
    <1,5>
];
```

**Code Sample 3.2** OPL Data for the Car Sequencing Example (`car.dat`)

## Setting Up the Project

You need to open a project file with its associated model and data files, and, optionally, load the data into the editor.

### Opening the Project File

The ILOG OPL Studio Main window should still be open from the previous tutorial. If not, you will need to launch it as described in *Launching ILOG OPL Studio* on page 18. When the Main window is open, you are ready to start the car sequencing tutorial by opening the project file.

Select Open>Project from the File menu, or click on the corresponding button. Then select `car.prj` from the list displayed in the `opl` directory.

ILOG OPL Studio then displays the project tree in the work space, showing the names of the associated model and data files in the tree structure. You can see that `car.mod` and `car.dat` are the files associated with `car.prj`. You will notice that the names `car.prj` and `car.mod` are highlighted in red. This is to remind you that `car.prj` is the active project, and thus, `car.mod` is the active model.

### Loading the Data

You can open the project without loading the data file into the editor. This feature is useful if you have a large data file that takes a long time to load. For this example we will load the data file by clicking on the name of the file with the right mouse button, then selecting the Load option, as shown in Figure 3.1.

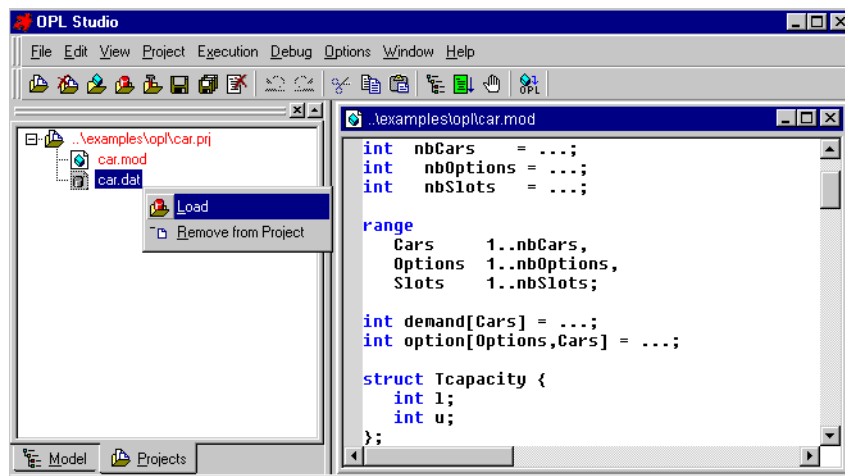
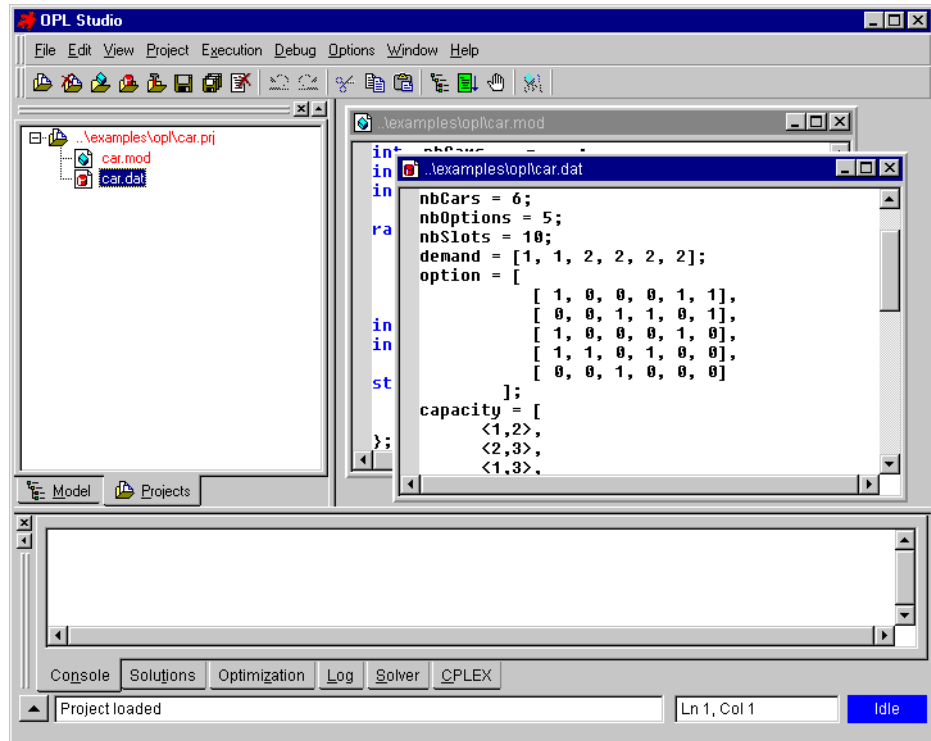


Figure 3.1 Loading a Data File for the Car Sequencing Example


ILOG OPL Studio has now opened the files associated with the project in the work space of the Main window. For this example, it has opened `car.mod` and `car.dat`.



*Figure 3.2 Main Window for the Car Sequencing Example*

## Executing the Project

Now that everything is set up in the project file, you can execute the project right away.

Click the Run button  in the tool bar of the Main window.

**Note:** *If you happen to have other model files, or other project files, open in the Main window, ILOG OPL Studio always gives precedence to the active project file. The most recently opened project model is by default the active project.*

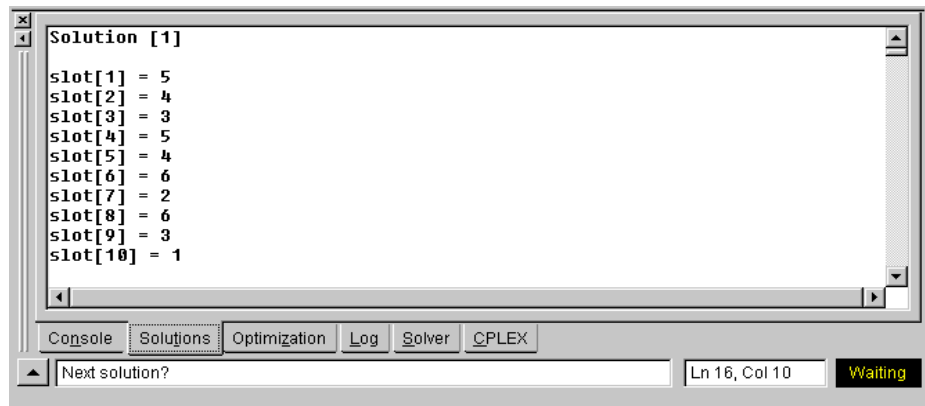
*You can set another project as the active one by right clicking on the new project name and selecting the option Set As Active Project.*

ILOG OPL Studio, when it executes a project:

- ◆ analyzes the project and produces the summary information in the model browser
- ◆ checks for syntactic or semantic errors
- ◆ executes the model displaying “OPL Studio is running” in the status bar, showing the name `car.prj` in the Path Name area, and changing the color patch to green
- ◆ displays the first solution in the Solutions notebook page in the Main window
- ◆ enters a waiting state when the first solution is found, displays “Next solution?” in the status bar, and changes the color patch to yellow.

## Examining the First Solution

While ILOG OPL Studio is in its waiting state, you can take a look at the first solution to the car sequencing problem. The `slot` variables are listed first. The `slot` variables specify which car is assigned to a given slot in the assembly line. You can see that slot 1 has car 5, slot 2 has car 4, and so on.



**Figure 3.3** `slot` Variable Results for Solution[1]

If you want to see another view of this result, go to the model browser and double-click the `slot` entry in the Variables folder. ILOG OPL Studio displays the following panel with the same result. Slot 1 (Index 1) has car 5, slot 2 (Index 2) has car 4, and so on.

Solution[1]: slot	
Indices	Data
1	5
2	4
3	3
4	5
5	4
6	6
7	2
8	6
9	3
10	1

**Figure 3.4** Alternative View of `slot` Variable Results for Solution[1]

If you scroll down through the Solutions notebook page, you see the results for the `setup` variables. The `setup` variables specify, given an option and a slot, whether the car assigned to the slot requires the option. From the first line, you can see that the car at option 1 and slot 1 [1,1] requires option 1 as indicated by the value 1. The car at option 1 and slot 2 [1,2] does not require option 1 as indicated by the value 0. You can scroll down through the list to see the assignments for all five options and the slots for each option.



**Figure 3.5** `setup` Variable Results for Solution[1]

Again, if you want to see another view of this result, go to the model browser and double-click the `setup` entry in the Variables folder. ILOG OPL Studio displays the following panel with the same results. This time the results are displayed in matrix format. The 5 options are listed on the left-hand side of the panel and the 10 slots appear across the top. Again, the values 1 and 0 in the matrix indicate whether the car at that intersection requires the option or not. The car at [1,1] requires option 1; the car at [1,2] does not require option 1.

Solution[1]: setup										
	1	2	3	4	5	6	7	8	9	10
1	1	0	0	1	0	1	0	1	0	1
2	0	1	1	0	1	1	0	1	1	0
3	1	0	0	1	0	0	0	0	0	1
4	0	1	0	0	1	0	1	0	0	1
5	0	0	1	0	0	0	0	0	1	0

**Figure 3.6** Alternative View of `setup` Variable Results for Solution[1]


## Copying the Results Matrix to a Spreadsheet

You can copy the results matrix from OPL Studio to an external spreadsheet such as Microsoft Excel, StarOffice or gnumeric.

Select a matrix or sub-matrix by dragging the mouse from the top left corner of the matrix to the bottom right corner. Press Ctrl+C to copy to the clipboard, then Ctrl+V to paste into the spreadsheet. Tabulation characters are placed between the cells as separators.

**Note:** You can read and write to an Excel sheet via OPL or OPL Script by using the keywords `SheetConnection`, `SheetRead` and `SheetWrite`.

## Continuing the Execution

While you have been looking at the first solution, ILOG OPL Studio has been in its waiting state. To produce all the remaining solutions to the problem, click the Continue Run button  in the tool bar.

ILOG OPL Studio continues its computation of subsequent solutions without stopping until it has found all the solutions. While it is running, it again displays “OPL Studio is running” in the status bar and the color patch changes to green.

When it completes its execution, ILOG OPL Studio returns to its idle state and the following occur:

- ◆ the status bar displays “OPL Studio is idle: 6 solution(s) found”
- ◆ the color patch returns to blue
- ◆ the model browser collapses and no longer shows the data structures.

You can look at the Solutions notebook to see all the solutions. As ILOG OPL Studio returns additional results, it adds them, one after the other, to the Solutions notebook page. Solution [6] is now visible, but you can use the scroll bar to see the other solutions.

Take a look at the Solver notebook page to see the statistics generated for the execution of this problem.


If you look at the Optimization notebook page, you see that it is empty. This model does not contain an optimization statement, so therefore this page remains empty.



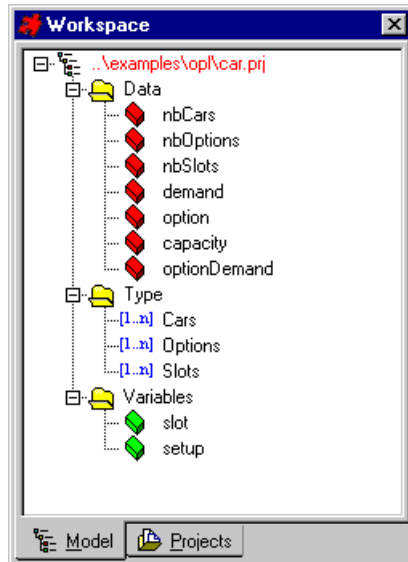
## Looking at the Model Structure

At this point in the tutorial, the data structure for `car.prj` is not displayed in the model browser. However, ILOG OPL Studio allows you to rebuild the data structure of a model without having to execute the model.

There are four ways of rebuilding the browser information. These are described in *Using the Model Browser* on page 67.

For example, click on the Rebuild Browser Information button  in the tool bar of the Main window.

ILOG OPL Studio builds the tree representing the data structures of the model in the model browser, just as it did when it executed the project.



**Figure 3.7** Model Browser for `car.prj` (Floating State)


The navigation tools of the model browser are available for your use at this time. When you click on an entry, ILOG OPL Studio highlights the line where the first occurrence of the entry is found in the edited model file.

**Note:** Because ILOG OPL Studio is in an idle state at this point, a double-click on an entry has no effect. However, when ILOG OPL Studio is in a waiting state during execution, you can double-click on an entry to display additional views of the results of a solution.

## Using Dynamic Display with ILOG OPL Studio

As the last task in this tutorial, let's take a look at the dynamic display feature of ILOG OPL Studio. Dynamic display plays an essential role in the development process, as it can bring important insights into the actual behavior of the various constraint solutions. Such insights can suggest different search strategies or the addition of redundant constraints.

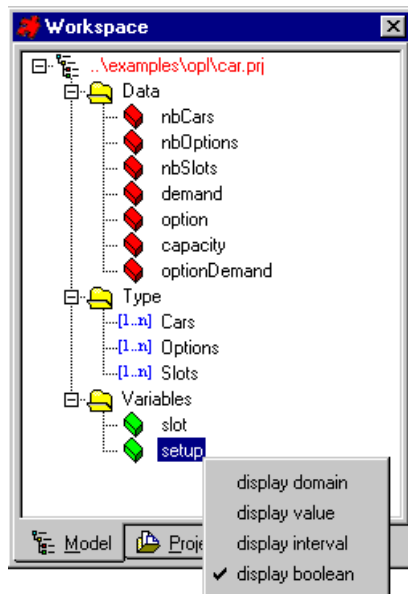
The variables of a model can be subject to dynamic display. You must first select the display option.

To see how to use the dynamic display feature in ILOG OPL Studio, go to the model browser. The tree structure of `car.prj` should still be displayed from the previous section. If not, click the Rebuild Browser Information button  in the tool bar, or select Execution>Browse Active Model from the menu bar.

Let's look at how ILOG OPL Studio handles the dynamic display of the `setup` variable. Click on `setup` in the Variables folder, then, with the cursor on `setup`, click the right mouse button. A pop-up menu appears with a list of the available display options. Specify your selection by checking one of the options.


- ◆ **display domain** – each variable is associated with an array of cells. Each cell in the array corresponds to a value in the domain of the variable. The cell is colored blue or beige based on the presence of that value in the domain.
- ◆ **display value** – each variable is associated with an array of cells. Each cell in the array corresponds to a value of the variable. The cell is colored blue when the value is assigned.
- ◆ **display interval** – each variable is mapped to an interval showing the lower bound and the upper bound. The execution narrows the interval. Holes in the domain are not shown.
- ◆ **display boolean** – each variable is mapped to a colored cell. This is useful for a variable whose domain is  $\{0,1\}$ . The cell becomes blue if the domain is reduced to  $\{1\}$  and beige if the domain is reduced to  $\{0\}$ .

For this example, select “display boolean”. A check mark appears to the left of the option, indicating that it is selected, as shown in Figure 3.8.



**Figure 3.8** Selecting a Dynamic Display Option

Now, execute the project again to see the dynamic display.


Click the Run button  in the tool bar.


This time, as it executes, ILOG OPL Studio opens a setup notebook page in the output area of the Main window. It displays the `setup` results in a matrix. Because you selected “display boolean” from the pop-up menu, each Boolean is displayed in a colored cell. During the search for a solution, the matrix is updated in real time. The cells appear gray or colored depending on the value the corresponding Boolean receives. A gray cell is associated with a Boolean variable that is still unassigned. A blue cell is true and a beige cell is false.

The following screen shows how the setup notebook page looks after the first solution is found and ILOG OPL Studio is in its waiting state.



**Figure 3.9** Setup Notebook Page Showing Boolean Display


Click on the Next button  and watch as ILOG OPL Studio computes the second solution.

Continue clicking on the Next button to see more solutions of the model displayed. To complete the execution, click on Continue Run .

**Note:** In dynamic display mode, the execution may be slower.

## Closing the Project

This completes the car sequencing example. To close the files associated with a project, you have to close the project. This is done in one of three ways:

- ◆ Select File>Close Active Project from the menu bar
- ◆ Click on the Close Active Project button  in the tool bar
- ◆ Right click on the project name in the work space, and select Close Project.

The project file and its associated files are saved and closed.

## ***Tutorial: Examining the Solution to a Scheduling Problem***

In this chapter, you will see how ILOG OPL Studio handles a scheduling model and uses Gantt charts to display results.

Using the house building example, you will:

- ◆ open the model file
- ◆ execute the model
- ◆ examine the solution of the problem, noting how ILOG OPL Studio handles the display of the activities and resources of a scheduling problem
- ◆ close the model file.

For this part of the tutorial, you will be using the file `house2.mod` from your release distribution. If you used the default directories at installation time, you can find this file at the following location:

- ◆ For UNIX systems  
`<installation-directory>/OPLSt37/examples/opl/scheduler`
- ◆ For Windows XP, Windows 2000, Windows NT 4, and Windows 98  
`c:\ILOG\OPLSt37\examples\opl\scheduler`

## The House Building Example

This example centers around the activities necessary to build a house and the budget constraints for each of these tasks. The house building example shows how OPL and ILOG OPL Studio support scheduling applications through the concepts of activities and resources.

In OPL, an activity can be thought of as an object containing three data items: a starting date, a duration, and an ending date. In the example, activities are the tasks needed to build a house.

OPL supports a variety of resources including unary, discrete, discrete energy, state and reservoirs. In this example, you will see the support of discrete resources. A discrete resource is a resource with a capacity. The capacity, which may vary over time, represents the number of available instances of the resource. In this example, a discrete resource is used to model the budget for building the house.

In planning the construction of the house, there are a certain number of activities that must be performed, each with a certain duration, and some that must be completed before others can be started. The table below lists each activity, its duration in days, and the activities that must precede it.

Name	Duration	Preceding Activities
masonry	7	
carpentry	3	masonry
plumbing	8	masonry
ceiling	3	masonry
roofing	1	carpentry
painting	2	ceiling
windows	1	roofing
facade	2	roofing, plumbing
garden	1	roofing, plumbing
moving	1	windows, facade, garden, painting

In addition to the time constraints, there are constraints concerning the budget for this house. Each activity requires the payment of an amount of money proportional to the duration of the activity. This amount, to be paid at the beginning of the activity, is set to \$1,000 per day. The total budget is \$29,000, which is \$1,000 times the sum of the duration of the project activities. Only \$20,000 is available at the beginning of the project; the remaining \$9,000 is

available 15 days after the beginning of the project. The goal is to minimize the duration of the project while taking into account both the time and budget constraints.

Code Sample 4.1 shows the OPL model for this example. This model is found in the file `house2.mod` of your release distribution. In this example, the data for the problem is contained within the model. It has not been separated out into a `.dat` file.

```
enum Tasks
{ masonry,carpentry,plumbing,
  ceiling,roofing,painting,
  windows,facade,garden,moving };

int duration[Tasks] = [7,3,8,3,1,2,1,2,1,1];

scheduleHorizon = 30;
Activity a[t in Tasks](duration[t]);

DiscreteResource budget(29000);

minimize
  a[moving].end
subject to {
  a[masonry] precedes a[carpentry];
  a[masonry] precedes a[plumbing];
  a[masonry] precedes a[ceiling];
  a[carpentry] precedes a[painting];
  a[ceiling] precedes a[painting];
  a[roofing] precedes a[windows];
  a[roofing] precedes a[facade];
  a[plumbing] precedes a[facade];
  a[roofing] precedes a[garden];
  a[plumbing] precedes a[garden];
  a[windows] precedes a[moving];
  a[facade] precedes a[moving];
  a[garden] precedes a[moving];
  a[painting] precedes a[moving];


  capacityMax(budget,0,15,20000);

  forall(t in Tasks)
    a[t] consumes(1000*duration[t]) budget;
};
```

**Code Sample 4.1** OPL Model for the House Building Example (`house2.mod`)

## Opening the Model File


If the Main window is not open, you will need to launch OPL Studio, as described in *Launching ILOG OPL Studio* on page 18. When the Main window is open, you are ready to start the house building tutorial by opening the model file. Select `Open>Model` from the File

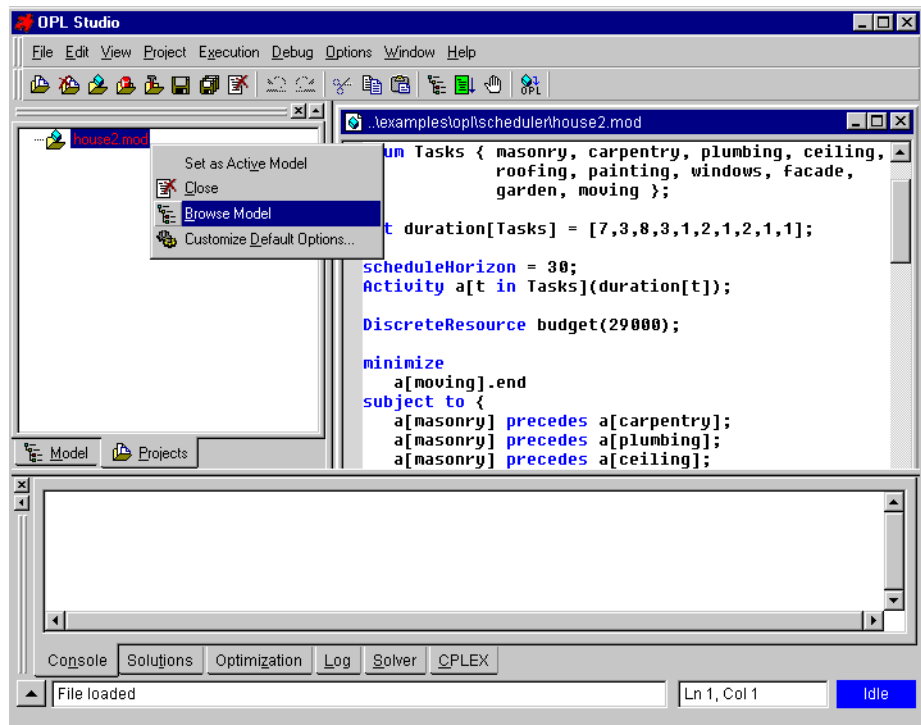
menu, or click on the Load Model button  in the tool bar, and select `house2.mod` from the `scheduler` directory.

ILOG OPL Studio then displays the `house2.mod` file in the work space.

## Looking at the Model Structure

Before executing the model, let's take a look at the data structure of `house2.mod` to see what has been defined in this model.

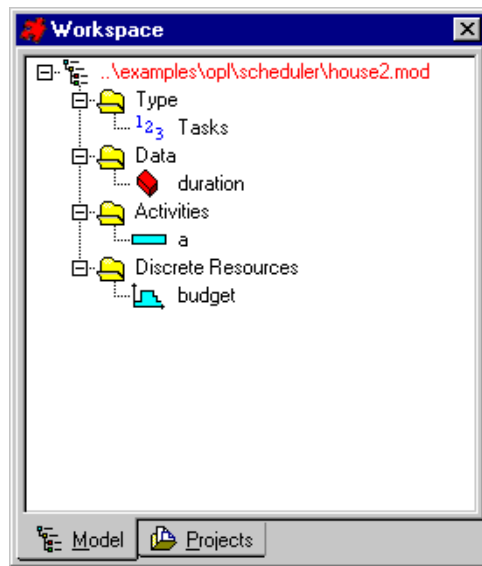
Right-click on the model name in the Projects window, then select  Browse Model, as shown in Figure 4.1.



**Figure 4.1** Main Window for the House Building Example



ILOG OPL Studio opens the Model browser and builds the tree of the model objects.



**Figure 4.2** Model Browser for house2.mod (Floating State)

As you can see, this model has the following objects: Type (Tasks), Data (duration), Activities (a), and Discrete Resources (budget).


For this model, you will want to pay particular attention to how ILOG OPL Studio displays the results of the activities and the discrete resources.

---

## Executing the Model

For this example, there is only a model file. The data has not been isolated in a `.dat` file, so therefore a project file (`.prj`) and a data file (`.dat`) do not exist. You are going to be executing only the model file (`house2.mod`).

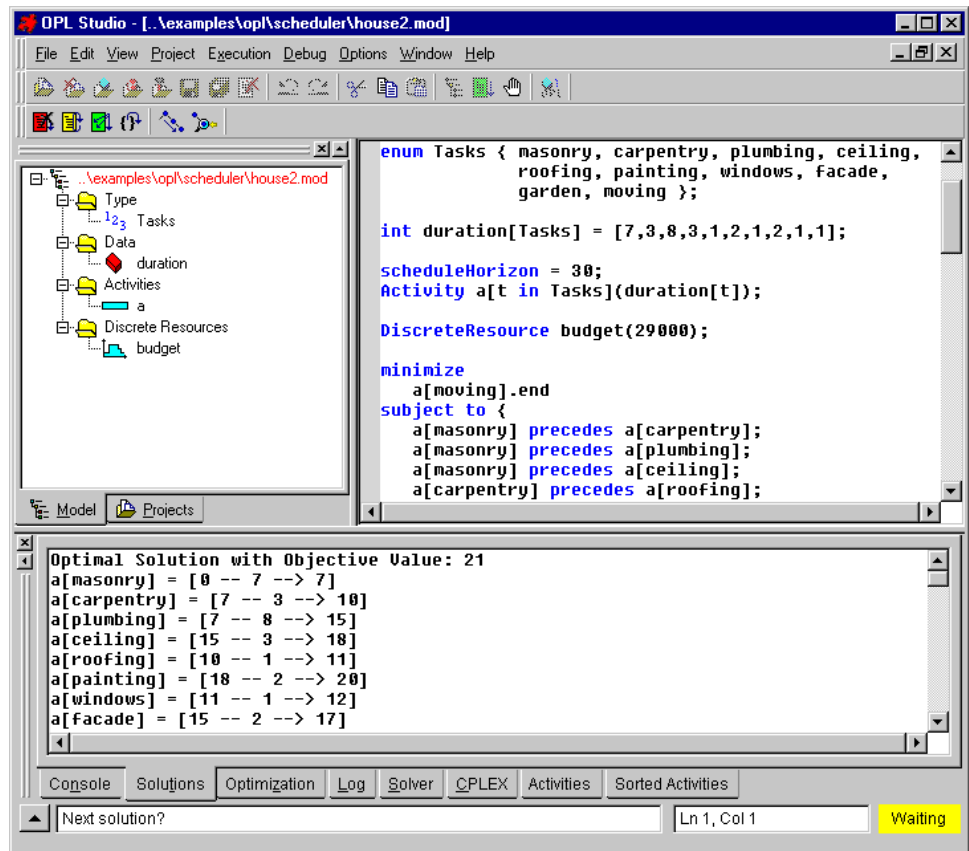
**Note:** Make sure that `house2.mod` is the active model. The name should appear in red on the Projects notebook page. If this is not the case, right-click on the name and select *Set As Active Model*.

Click the Run button  in the tool bar of the Main window.

In the same manner as when executing a project, ILOG OPL Studio:

- ◆ analyzes the model and, if not already done, produces the summary information and updates the Model browser
- ◆ checks for syntactic or semantic errors
- ◆ executes the model displaying “ILOG OPL Studio is running” in the status bar, showing the name `house2.mod` in the Path Name area, and changing the color patch to green
- ◆ displays the results in the notebook pages of the output window
- ◆ enters a waiting state when the optimal solution is found, displays “Next solution?” in the status bar, and changes the color patch to yellow.

The Main window now looks like this.



**Figure 4.3** Main Window with house2.mod Solution

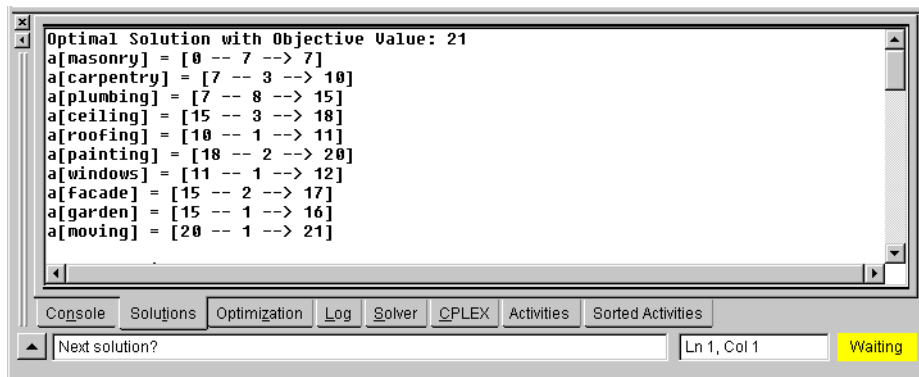
Note that because the model file contains an optimization statement (`minimize`), ILOG OPL Studio has found the optimal solution to the problem. While OPL Studio is in the waiting state, and before you continue, you have the opportunity to examine the results.

## Examining the Solution

The results of Activities and Resources are the important things to look at in this example. Let's first look at the Activities results and see how ILOG OPL Studio displays those results. Then, let's look at how ILOG OPL Studio displays the results for Resources.

### Looking at the Activities Results

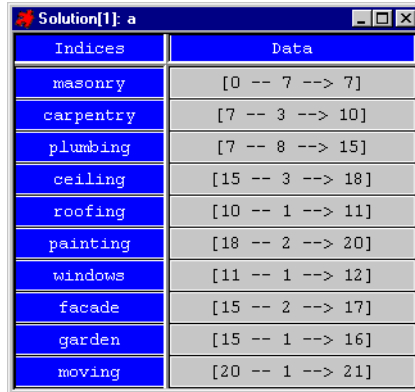
First, go to the Solutions notebook page. The first line shows that the duration of the project, taking into account the precedence and budget constraints, is 21 days. The next part of the solution shows the schedule for the tasks. These are the results for the Activities of the problem and are shown as an activity vector. The data is in the format of  $s \text{ -- } d \text{ --> } e$ . The activity starts at time  $s$  for a duration of  $d$  units of time, and is completed at time  $e$ . For example, you can see that masonry begins at time 0, has a duration of 7 units, and is completed at time 7. Carpentry begins at time 7, has a duration of 3 units, and is completed at time 10.



**Figure 4.4** Activities Results for house2.mod Example

## Using the Model Browser

To get another view of the results of the Activities, go to the Model browser and double-click the a entry in the Activities folder. ILOG OPL Studio first displays the following panel that essentially shows the results in the same format as shown in the Solutions notebook page. You will need to move the Gantt chart, as it is displayed on top of the activities results.



Indices	Data
masonry	[0 -- 7 --> 7]
carpentry	[7 -- 3 --> 10]
plumbing	[7 -- 8 --> 15]
ceiling	[15 -- 3 --> 18]
roofing	[10 -- 1 --> 11]
painting	[18 -- 2 --> 20]
windows	[11 -- 1 --> 12]
facade	[15 -- 2 --> 17]
garden	[15 -- 1 --> 16]
moving	[20 -- 1 --> 21]

**Figure 4.5** Alternative View of Activities Results for house2.mod Example

ILOG OPL Studio also displays a Gantt chart representation of the activities. The horizontal scale shows the total duration of the project, that is 21 units. The vertical scale on the left side of the window lists all the activities for the project. The center area shows the starting and ending dates of each task and the precedence relationships among the tasks. You can see that masonry begins at day 0, has a duration of 7 days, and ends on day 7. Carpentry begins on day 7, has a duration of 3 days, and ends on day 10.

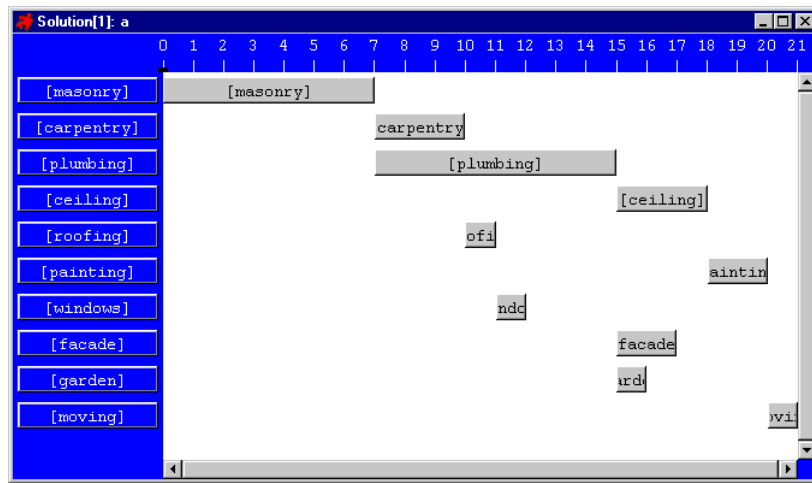


Figure 4.6 Gantt Chart of Activities Results for house2.mod Example

### Using the Gantt Chart

The Gantt chart can display either activities alone, or activities and resource allocation.

- ◆ Activities are displayed:
  - in the Activities and Sorted Activities output area
  - or, by double clicking on Activities items in the model browser.
- ◆ Activities and resource allocation are displayed for unary resources only, by double clicking on the UnaryResources items in the model browser.

**Note:** Before using the keyboard with the Gantt chart, click on the chart in order to give it the keyboard focus. In the Activities and Sorted Activities notebook pages, if the cursor is outside the chart, the *z* key will take you to the Optimization tab.

- ◆ To zoom

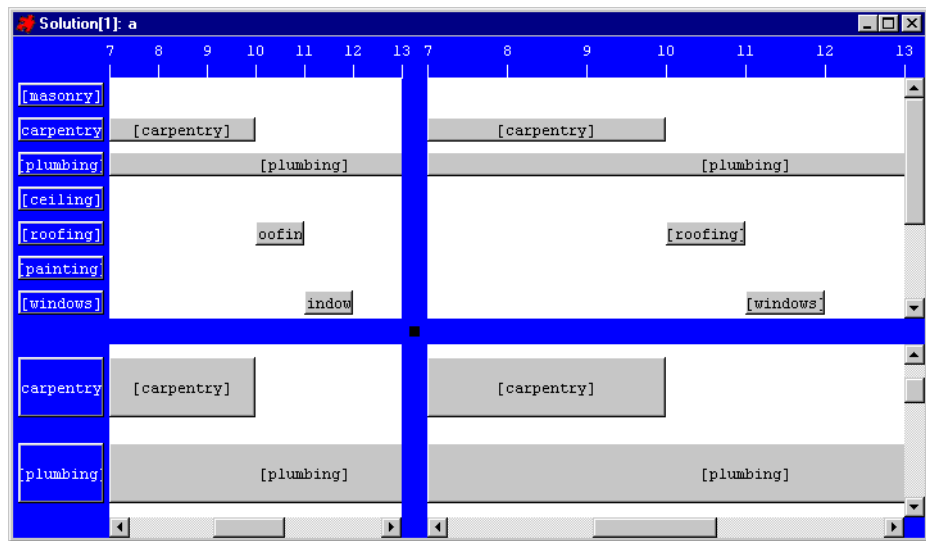
You can zoom in and zoom out of the chart using the keyboard or the mouse.

- To zoom in on the time axis there are two possibilities:
  - press the two keys: Shift + z
  - or select a time zone by left-clicking and dragging on the horizontal axis
- To zoom out of the time axis:
  - press the two keys: Shift + u

- To zoom in on the vertical axis, there are two possibilities:
  - press the z key
  - or select a vertical zone by left-clicking and dragging on the vertical axis
- To zoom out of the vertical axis:
  - press the u key
- ◆ To scroll
  - Use the horizontal scroll bars or the arrow keys → and ← for time scrolling.
  - Use the vertical scroll bars or the arrow keys ↓ and ↑ for vertical scrolling.
- ◆ To fit the contents to the frame, or return to the initial state
  - To fit the contents on the time axis, press the two keys: Shift + f.
  - To fit the contents on the vertical axis, press the f key.
  - To return to the initial state (no zoom) press the two keys: Shift + i.
- ◆ To split the chart into four views

**Note:** Before using the keyboard on one of the four views, click on the view in order to give it the keyboard focus.

If the Gantt chart is large, you cannot zoom in and see activities that are far apart. In order to see them together, you can use the split feature. By dragging the black square located at the top left corner of the chart you obtain four views. Figure 4.7 contains a Gantt chart split into four views.



**Figure 4.7** Gantt Chart Split into Four Views

The top left and top right views are constrained by their vertical scale, which means that these two views display the same row, but you can select different zoom factors or scroll states along the time axis. The same applies to the bottom left and bottom right views.

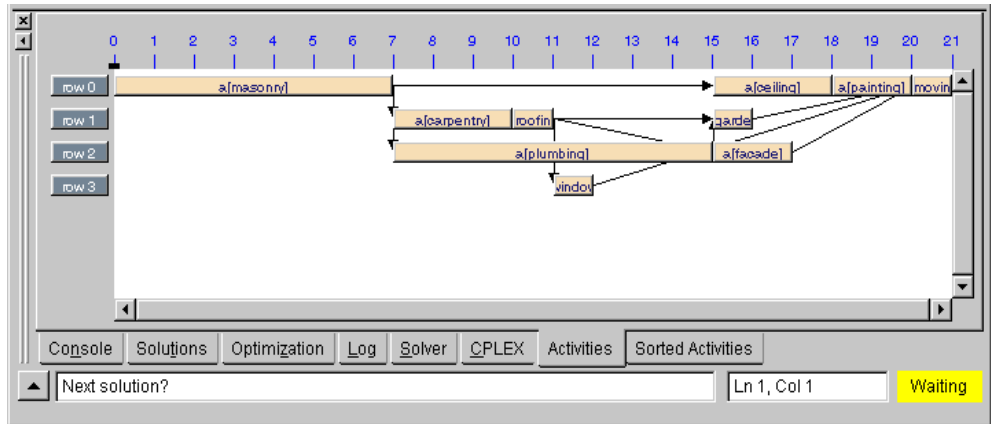
The top left and bottom left views are constrained by their horizontal scale (time axis), but you can select different zoom factors or scroll states for the vertical axis. The same applies to the top right and bottom right views.



### The Activities Notebook Page

ILOG OPL Studio provides another way to look at the Activities results. During the execution of the model, ILOG OPL Studio added two notebook pages, Activities and Sorted Activities, to the display area. These notebook pages provide additional views of the activities results.

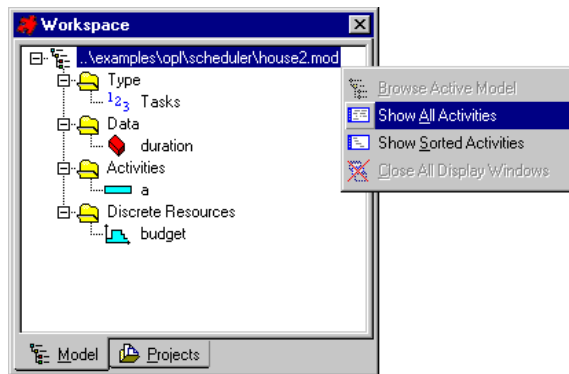
Click on the Activities tab to bring the Activities notebook page to the foreground.



**Figure 4.8** Activities Notebook Page for house2.mod Example

The chart displayed in the Activities notebook page again shows the time line on the horizontal scale with a duration of 21 days. The center area shows each task with its starting and ending date. You can see that the precedence of the tasks is indicated by the arrows going from one task to another.

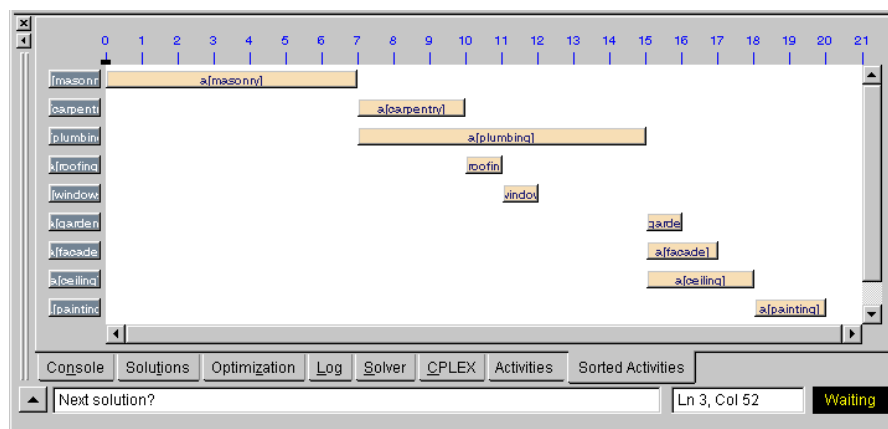
You can display the same information via the model browser. Right-click on the root item and select Show All Activities.



**Note:** The Activities notebook page is displayed by default when a model or project contains activities objects. If you do not want the Activities notebook page displayed, you can deselect this option in the Options dialog box (for the current project) or in the Default options dialog box (for all other projects). See Setting Output Options on page 169.


## The Sorted Activities Notebook Page

Click on the Sorted Activities tab to bring that notebook page to the foreground.



**Figure 4.9** Sorted Activities Gantt Chart for house2.mod Example

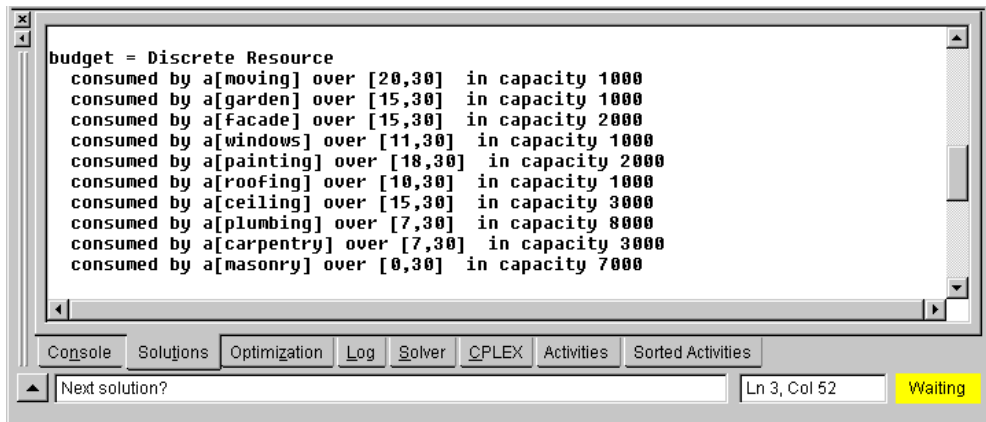
This notebook page displays all the activities in sorted order. Again, the horizontal scale shows the duration of the project, 21 days. The vertical scale lists all the activities, this time sorting them by precedence. In the center, you can again see the chart of the activities with their precedence.

You can display the same information via the model browser by right-clicking on the root item and selecting  Show Sorted Activities.

**Note:** The Sorted Activities notebook page is displayed by default when a model or project contains activities objects. If you do not want the Sorted Activities notebook page displayed, you can deselect this option in the Options dialog box (for the current project) or in the Default options dialog box (for all other projects). See Setting Output Options on page 169.

## Looking at the Resources Results

To see the results for the resources defined in the model, you need to go back to the Solutions notebook page. Click the Solutions tab to bring it to the foreground. Then use the vertical scroll bar to move down through the solution until you come to `budget = Discrete Resource`.

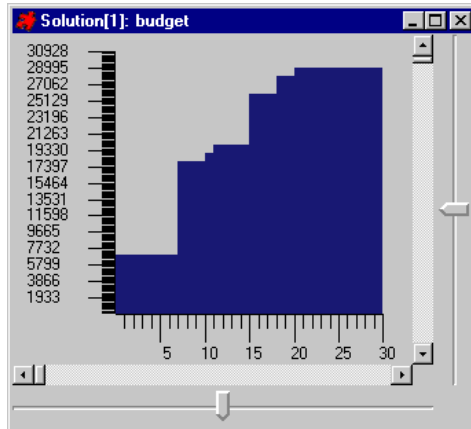


**Figure 4.10** Discrete Resource Results for house2.mod Example

The results show how the budget resources for each task are consumed for the life of the project. The tasks are listed in reverse order of their precedence. Go to the last line to look at the masonry budget resources. This line shows that masonry consumes \$7,000 (\$1,000 for each day of its 7-day duration). The [0,30] indicates that the resource is consumed from day 0, which is the start time of the masonry task, to day 30, which is the end of the project. The value of 30 comes from the `scheduleHorizon = 30` statement in the model file (see Code Sample 4.1). The carpentry task consumes \$3000 of the budget, beginning at day 7 and

ending at day 30. In this way, you can see the consumption of budget resources for all the tasks of the project.

To get another view of these results, go to the Model browser and double-click the `budget` entry in the Discrete Resources folder. ILOG OPL Studio displays the results in a bar chart (see Figure 4.11).



**Figure 4.11** Alternative View of Discrete Resources Results for `house2.mod` Example

Discrete Resources are displayed with a bar chart in ILOG OPL Studio. The horizontal scale shows the time line, just as it did for the Gantt chart. The vertical scale represents the maximum number of units (in this case, dollars) used at any point in time throughout the duration of the project. The bars in the chart give the evolution of the resource over time.

This panel has two sliders that you can use to adjust the scaling factor along the respective axes. When this panel comes up for the first time, ILOG OPL Studio uses a scaling factor to guarantee that the whole chart appears in the viewing area. You may want to adjust the scaling factor to see more details of the results and you can do this with the sliders. Then, use the scroll bars to bring any part of the chart into the viewing area.

**Note:** The graphic representation of unary resources is different. Double-clicking on a unary resource displays a Gantt chart with resource allocation. See `bridge.prj` for an example.

### The Optimization Notebook Page

In the previous two examples in Chapters 2 and 3, the Optimization notebook page was empty. When a model contains an optimization statement, as this one does (the minimize statement in Code Sample 4.1), this notebook page contains information.

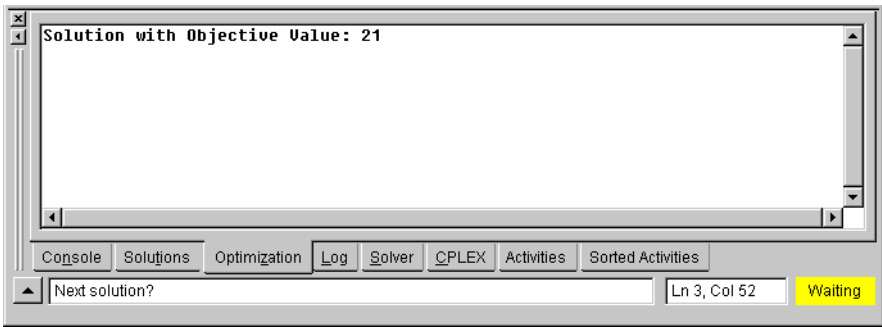


Figure 4.12 Optimization Notebook Page for house2.mod Example

The Optimization page displays the optimal solution.

### The Solver Notebook Page

As with the other examples, the Solver notebook page contains statistics generated by ILOG OPL Studio during the execution of the model.

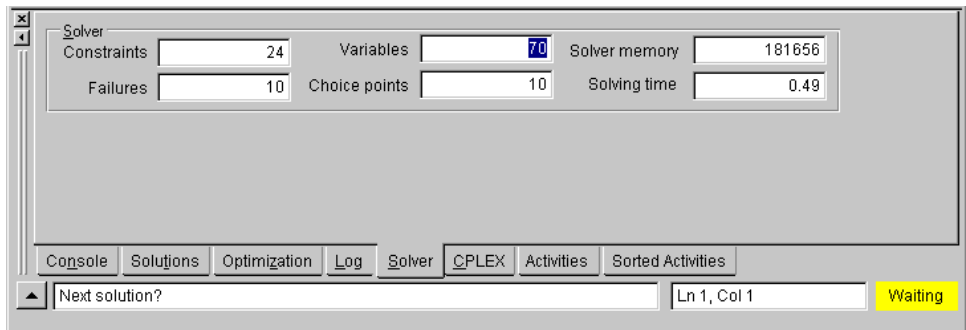



Figure 4.13 Solver Notebook Page for house2.mod Example

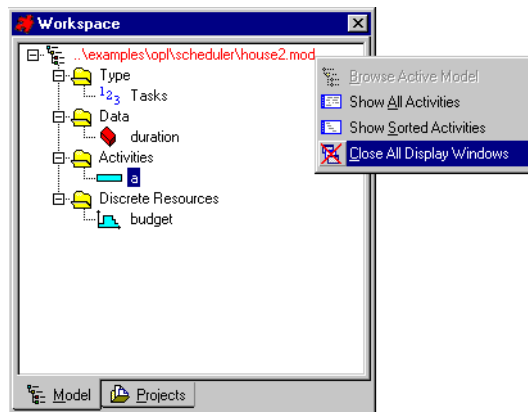
## Completing the Execution

While you have been examining the results after running the model file, ILOG OPL Studio has been in a waiting state. You can now complete the execution.

Click the Abort button  in the execution tool bar of the Main window.

As it completes the execution, ILOG OPL Studio closes all the panels that were opened from the Model browser. However, the solutions and results remain in the Solutions, Optimization, Activities, and Sorted Activities notebook pages.

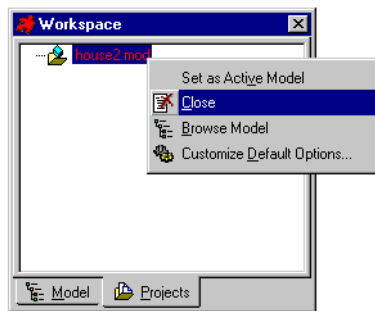
You can also close all the panels by right-clicking on the root item in the model browser.



## Closing the Model File

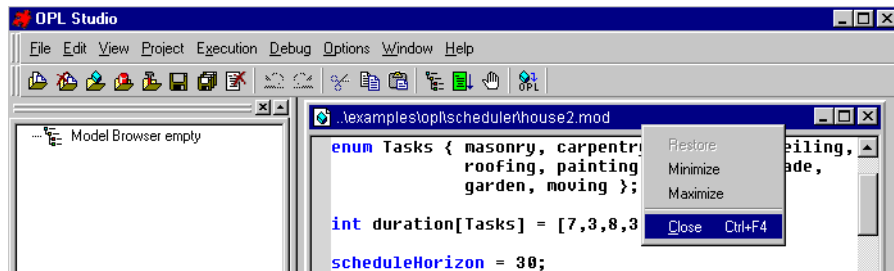
This completes the house building example. You can close the model file in one of the following ways:

- ◆ Select Close Current Editor from the File menu
- ◆ Press the keys Ctrl + F4
- ◆ In the Projects notebook page, right-click on the active model (house2.mod) and select Close from the menu, as shown in Figure 4.14.



**Figure 4.14** Closing a Model in the Projects Page

- ◆ In the editing area (when the window is in cascade mode) select Close from the menu, or click on the button with an X in it, as shown in Figure 4.15.



**Figure 4.15** Closing a Model in the Editing Area



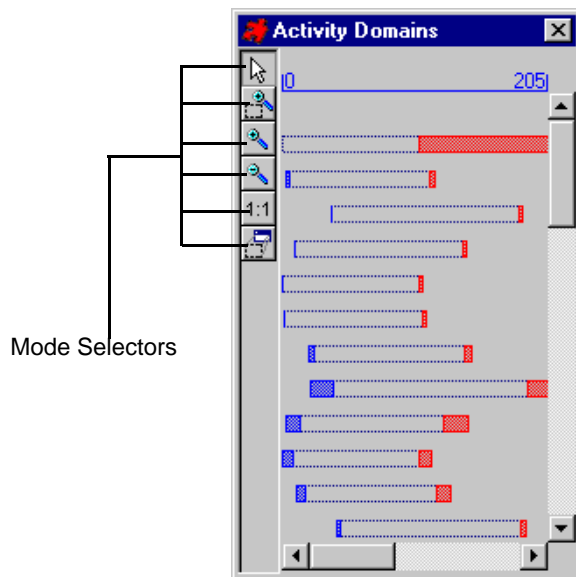


## ***Tutorial: Scheduling-Specific Dynamic Display***

In this chapter you will see how to use the Activity Domains window.

OPL Studio enables you to define scheduling-specific dynamic display. The Activity Domains dockable window shows the algorithm reducing the domains of the activities during a search.

## The Activity Domains Window



The vertical bar to the left of the Activity Domains window contains mode selectors. First click a selector button, then click in the main part of the window for the selected mode to become effective.

## Mode Selectors

- **Time selector**



Click on the arrow button, then click on the beginning of the activity and drag the cursor across the activity, holding down the left mouse button. An approximate value of the time at a given location appears in the center of the time scale.

- **Elastic zoom mode**



First, adjust the Activity Domains window to the required size. Click on the zoom button then, with the left mouse button, trace a rectangle around the element(s) you want to enlarge. When you release the mouse button, the enlarged image fills the window.

- **Zoom in**



Click on the button with a + sign, then click on the image in the main window. The image is increased by a factor of 2 each time you click.

- **Zoom out**



Click on the button with a – sign, then click on the image in the main window. The image is decreased by a factor of 2 each time you click.

- **View all activity domains**



Click on this button then click in the main part of the window. All the activity domains become visible; their sizes are adjusted to fit into the window.

- **Create new window**



Click on this button, then trace a rectangle around the element(s) you want to view in another window. A new dockable window is created, with a zoom level defined by the rectangle.

## The Bridge Example

Using the distributed project `bridgebr.prj` you can:

- ◆ browse the model and select the “display domain” option
- ◆ select a debug option, then execute the model
- ◆ look at the algorithm animation when searching for solutions.

If you used the default directories at installation time, you can find the `bridgebr.prj` file at the following location:

- ◆ For UNIX systems  
`<installation-directory>/OPLSt37/examples/opl/scheduler`
- ◆ For Windows XP, Windows 2000, Windows NT 4, and Windows 98  
`c:\ILOG\OPLSt37\examples\opl\scheduler`

### To select the “display domain” option

1. Open the `bridgebr.prj` file.
2. Browse the model.
3. Right-click on activity `a` and select the option "display domain".

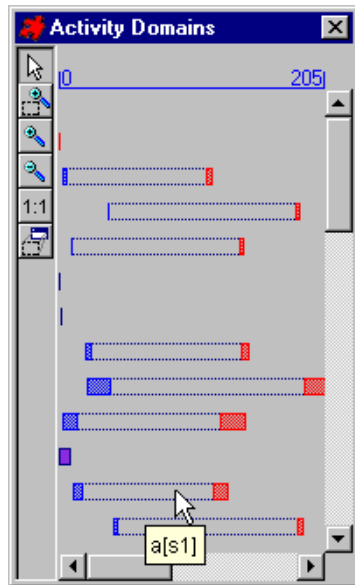
### To execute the model

1. Select: Debug>Stop at Choice Point.
2. Click Run.


### To observe the algorithm animation

1. The activity domains appear after the initial propagation, then the search begins.
2. Click 5 times on Next.

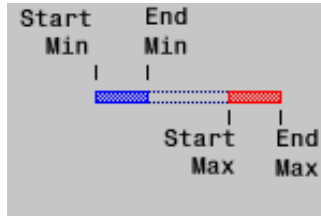
- Place the mouse pointer over the 11th task. A tooltip appears with the activity name `a[s1]`.



#### To zoom in on the `a[s1]` activity

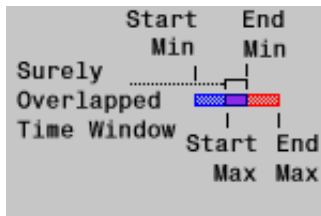
- Enlarge the Activity Domains window to the required size.
- Click on the zoom mode selector  in the vertical tool bar.
- Place the cursor in the main part of the window, hold down the left mouse button and trace a rectangle around the `a[s1]` activity. Then lift your finger off the mouse button.

4. The `a[s1]` activity fills the enlarged window.



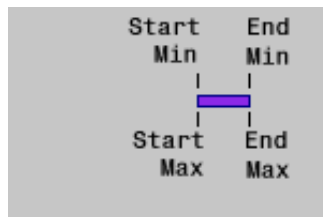
Because of the breakable nature of some of the activities in this example, the Start Min-End Min blue rectangle and the Start Max-End Max red rectangle may not always be the same size.

5. Click Next a 6th time.



As soon as `end min` passes `start max`, the time window between these two dates is the surely overlapped time window.

6. Click Next a 7th time. The activity is totally instantiated.



When `start min = start max` and `end min = end max`, the activity schedule is totally determined.

7. Click on Continue Run to see the algorithm finding solutions and backtracking.

**Reminder:** You can change the colors of the time windows in the Advanced notebook page of the Options dialog box.

## ***Tutorial: User-Defined Dynamic Display***

In this chapter you will see how to use the Drawing Board. OPL Studio enables you to define your own backtrackable 2D graphical representation using the Drawing Board to animate the search algorithm.

---

### **The Drawing Board**

The Drawing Board is a dockable window that you can float and resize.

**Reminder:** *To float or dock a window, see Dockable GUI Elements on page 33.*

The Drawing Board creates ellipses, arrows, lines, arcs, polygons, polylines, grids and labels. You can find the description of the `Board` keyword:

- ◆ in the online help (select `Board` and press the F1 key)
- ◆ in the *ILOG OPL Studio: Language Manual*.

## The Square Example

Using the distributed model `squaregr.mod` you can:

- ◆ execute the model
- ◆ examine the first solution
- ◆ look at the algorithm animation when searching for the remaining solutions.

If you used the default directories at installation time, you can find the `squaregr.mod` file at the following location:

- ◆ For UNIX systems  
`<installation-directory>/OPLSt37/examples/opl/scheduler`

- ◆ For Windows XP, Windows 2000, Windows NT 4, and Windows 98

`c:\ILOG\OPLSt37\examples\opl\scheduler`

The aim of the square example is to place a set of small squares of different sizes into a large square. The model is solved using scheduling algorithms, but the Gantt chart representation used by default for scheduling representation does not fit the natural representation of the problem. In our example we create a specific 2D representation by using the Drawing Board.

In the model we declare a color table containing standard X11 names:

```
string colors[1..21] = [
    "red", "green", "blue", "yellow", "pink",
    "brown", "magenta", "cyan", "white", "black",
    "turquoise1", "SeaGreen1", "gold1", "IndianRed1", "Siennal",
    "tan1", "salmon1", "orange1", "tomato1", "HotPink1",
    "orchid2"
];
```

Next we declare the Drawing Board. Note that you can declare more than one if you need to.

```
Board b;
```

In the search part of the model we use Drawing Board methods to draw. We can bind data or variables to the method parameters;

```
search {
    // Drawing the large square
    b.rectangle(0,0,112,112,2);
    // Drawing the small colored squares using variables and data
    forall(i in Squares) {
        b.filledRectangle(x[i].start,y[i].start,size[i],size[i],1,colors[i]);
        b.rectangle(x[i].start,y[i].start,size[i],size[i],1);
    };
    setTimes(x);
    setTimes(y);
};
```

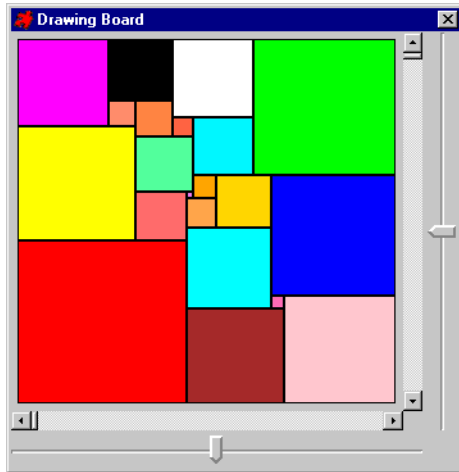


For instance the large square, which is pure constant data, is drawn using the following instruction:

```
b.rectangle(0,0,112,112,2);
```

The small squares, whose locations depend upon variable values, use two variables for (x, y) location and data for height, width and colors.

Load the model `squaregr.mod` into OPL Studio, then click on Run to execute it.



By clicking on Continue Run, you will see the search for solutions. As soon as a location is found for a square (that is, its x and y are bounded variables) the square is drawn. When backtracking occurs, the square is cleared.

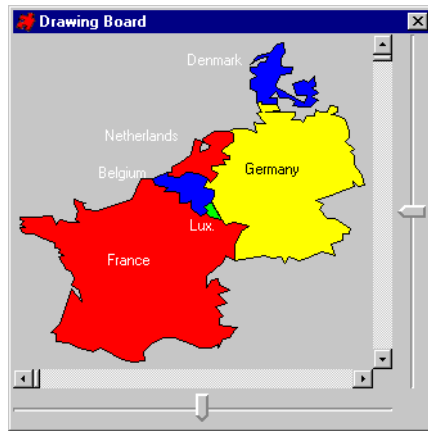
By default the drawing board is docked, but you can choose to have it automatically undocked. Before executing a model, uncheck the Docked option in the Advanced page of the Options dialog box (see page 171).

You can specify a width, a height and a scaling factor (between 0.0 and 2.0) when you initialize the drawing board. If the Docked box is checked, only the height/width ratio is guaranteed. If Docked is unchecked, the height and width values are respected.

## The Map Example

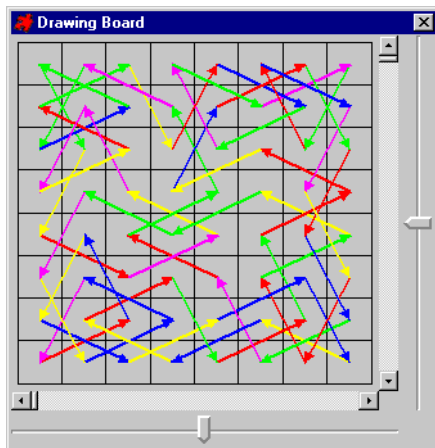
Another example of Drawing Board use, coloring a map, can be found in:

`<installation-directory>/OPLSt37/examples/opl/mapgr.prj`



## The Euler Example

Try also `examples/opl/eulergr.mod` to see another instance of Drawing Board use, this time involving a grid and colored arrows.



## Tutorial: Debugging the Search Strategy

In this chapter, you will see the debug feature of ILOG OPL Studio applied to the search procedure of constraint programming models. You will learn how to display the search tree and how to use it in conjunction with the Choice Stack, the Drawing Board, and other dynamic display facilities.

**Note:** *The debug feature primarily aims at tracing the search strategy of the model. Models that are deterministic (that is, linear programming models) have no search strategy and hence the debug feature has no effect.*

If you used the default directory at installation time, you will find the files you need at the following location:

- ◆ For Windows XP, Windows 2000, Windows NT 4, and Windows 98

`c:\ILOG\OPLSt37\examples\opl`

- ◆ For UNIX systems

`<installation-directory>/OPLSt37/examples/opl`

## A Basic Example with the Eight Queens Problem

There are some preliminary steps to take before executing the model. You need to modify the code and select the appropriate options.

### Setting up the Example

1. Open the file:

```
examples/opl/queens8.mod
```

This model contains the following code:

```
var int queens[1..8] in 1..8;
solve {
    forall(ordered i,j in 1..8) {
        queens[i] <> queens[j] ;
        queens[i] + i <> queens[j] + j;
        queens[i] - i <> queens[j] - j
    };
};
```

2. Add a simple search procedure:

```
search {
    forall(i in 1..8)
        tryall(v in 1..8)
            queens[i] = v;
};
```

3. From the Debug menu, select the options Stepping in Model and Display Search Tree.
4. Browse the model by selecting the menu item Execution>Browse Active Model. Right-click on the queens item in the Model notebook page and select "display domain".

### Executing the Model

1. Execute the model by clicking on the green Run button.
2. Open the Choice Stack and the Current Node Inspector (via the View menu). Click three times on the Next button. You will see the yellow arrow in the editor margin pointing to the line:

```
queens[i] = v;
```

Simultaneously the Current Node Inspector expands this choice as `queens[1] = 1.`

3. Click one more time on Next. You will see the result of the choice and of the constraint propagation on the chessboard.
4. Click one more time on Next. The next choice tried is `queens[2] = 1.`

Choice Points: 3   Failures: 0   Solutions: 0

```

var int queens[1..8] in 1..8;
solve {
  forall(ordered i,j in 1..8) {
    queens[i] <> queens[j] ;
    queens[i] + i <> queens[j] + j ;
    queens[i] - i <> queens[j] - j ;
  };
  search {
    forall(i in 1..8)
      tryall(v in 1..8)
        queens[i] = v;
  };
};

```

[0]--->solve  
 [0]--->forall(i in 1..8)  
 [0]--->tryall(v in 1..8)  
 [1]--->queens[1] = 1  
 [1]--->tryall(v in 1..8)  
 [2]--->queens[2] = 1

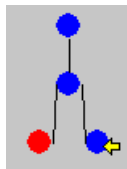
[2]--->queens[2] = 1

	1	2	3	4	5	6	7	8
[1]								
[2]								
[3]								
[4]								
[5]								
[6]								
[7]								
[8]								

Optimization   Log   Solver   CPLEX   queens

Because the value 1 has already been removed from the domain of the variable `queens[2]`, this second choice leads to a failure.

- Click once more on Next. The previous blue node is now red and a brother node, representing the choice of the value 2 for the same variable, is to be explored:



- Click Next two more times. You will notice that the value 2 has already been removed from the `queens[2]` variable, so this choice leads to a failure too. However, the value 3 is in the domain of the variable. The constraint propagation that follows does not lead to a failure. Other variables see their domain reduced by the propagation algorithm.

The screenshot displays the ILOG OPL Studio 3.7 interface for solving the Eight Queens problem. The top-left pane shows a search tree with 3 choice points (blue dots), 2 failures (red dots), and 0 solutions (green dots). The top-right pane contains the OPL code for the problem. The bottom-left pane shows the command window with the execution log. The bottom-right pane shows an 8x8 chessboard with the solution visualized: queens are placed at (1,1), (2,3), (3,5), (4,2), (5,7), (6,4), (7,6), and (8,8).

**Search Tree:**

- Choice Points: 3
- Failures: 2
- Solutions: 0

**OPL Code:**

```
var int queens[1..8] in 1..8;
solve {
  forall(ordered i,j in 1..8) {
    queens[i] <> queens[j];
    queens[i] + i <> queens[j] + j;
    queens[i] - i <> queens[j] - j;
  };
  search {
    forall(i in 1..8)
      tryall(v in 1..8)
        queens[i] = v;
  };
};
```

**Command Window:**

```
[0]--->solve
[0]--->forall(i in 1..8)
[0]--->tryall(v in 1..8)
[1]--->queens[1] = 1
[1]--->tryall(v in 1..8)
[2]--->queens[2] = 3
[2]--->tryall(v in 1..8)
[3]--->tryall(v in 1..8)
```

**Chessboard Visualization:**

	1	2	3	4	5	6	7	8
1	Queen							
2			Queen					
3					Queen			
4				Queen				
5						Queen		
6							Queen	
7								Queen
8								

**Buttons:** Optimization, Log, Solver, CPLEX, queens

## Continuing the Execution

To continue the execution step by step, click the Next button repeatedly and watch how ILOG OPL Studio assigns the variables. At this point, we describe:

- ◆ the tree mode selectors
- ◆ tree abstraction
- ◆ the default search strategy of ILOG OPL Studio.

## Search Tree Mode Selectors

### • Selection mode



Click on the arrow button, then double-click on a node belonging to the current branch.

### • Elastic zoom mode



Click on the zoom button then, with the left mouse button, trace a rectangle around the element(s) you want to enlarge. When you release the mouse button, the enlarged image fills the window.

### • Zoom in



Click on the button with a + sign, then click in the search tree window. The image is increased by a factor of 2 each time you click.

### • Zoom out



Click on the button with a – sign, then click in the search tree main window. The image is decreased by a factor of 2 each time you click.

### • View whole content



Click on this button then click in the search tree window. The whole tree becomes visible.

### • Create new window



Click on this button, then trace a rectangle around the element(s) you want to view in another window. A new dockable window is created, with a zoom level defined by the rectangle.

## Tree Abstraction

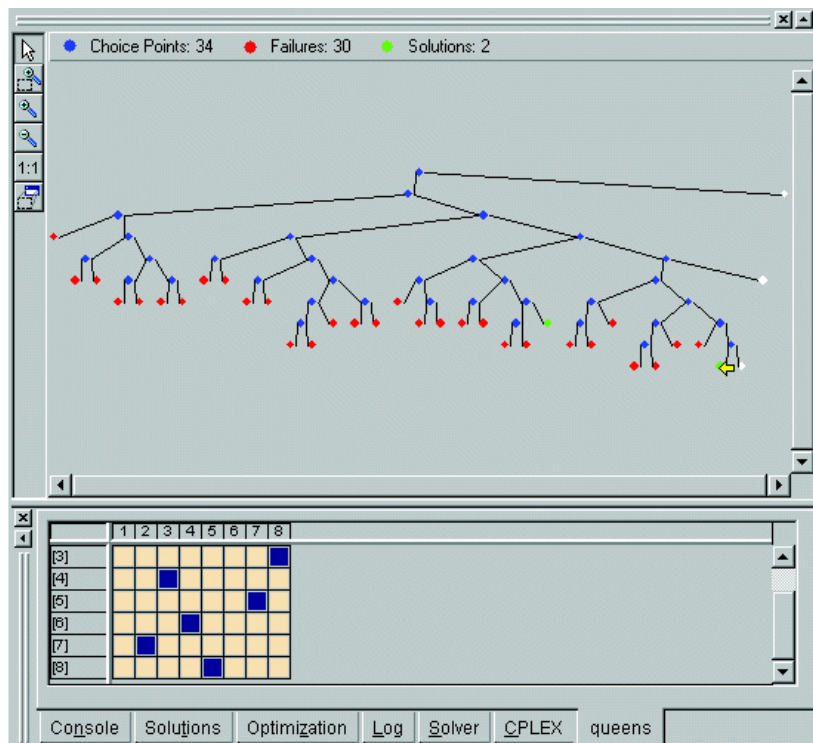
The search tree used internally by ILOG Solver is always a binary tree. ILOG OPL Studio abstracts this binary tree as an n-ary tree so that it resembles the search procedure you defined. This tree abstraction facility is available only if you have defined the choice points yourself using the `try` or `tryall` instructions.

## Default Search Strategy

Click on Abort and edit the model to remove the search procedure.

It is important to observe that, although the model now has no search strategy, the debug features are still beneficial. As explained in the *ILOG OPL Studio: Language Manual*, OPL has default search strategies that are used when no search strategy is specified. The debug features work, in fact, on these default search procedures, and OPL Studio stops at each instruction or at each choice point inside them. However, the search tree is not abstracted as an n-ary tree and the basic internal binary tree of ILOG Solver is displayed instead.

Select the menu option `Debug>Stop At Solution`. Rerun the model and click once on Next. You should obtain the following display:



**Figure 7.1** The Eight Queens Model with the Default Search Strategy



In the search tree:

- ◆ The red nodes are the failures.
- ◆ The green nodes are the solutions.
- ◆ The blue nodes are the explored choice points (i.e. the interior nodes).
- ◆ The white nodes are the nodes created internally by ILOG Solver and that remain unexplored so far.
- ◆ The black nodes are the unexplored nodes that ILOG Solver has finally pruned (visible only in Slice-Based Search).
- ◆ The root node, although blue, does not correspond to a choice point. When the yellow arrow points to the root, the algorithm is performing the initial domain reduction.

To end the 8 Queens example, click on Abort.

---

## The Frequency Allocation Example

Using the frequency allocation problem, you will:

- ◆ execute a project using the debug feature of ILOG OPL Studio
- ◆ try two different exploration search strategies
- ◆ visualize dynamically the search strategy in the search tree.

This example centers around the frequency allocation problem that appears in Chapter 14 of the *ILOG OPL Studio: Language Manual*.

The frequency allocation problem consists of allocating frequencies to a number of transmitters so that there is no interference between transmitters and the number of allocated frequencies is minimized. This problem is an actual cellular phone problem where the network is divided into cells. Each cell contains a number of transmitters whose locations are specified. The interference constraints are as follows:

- ◆ the distance between two transmitter frequencies within a cell must not be less than 16
- ◆ the distance between two transmitter frequencies from different cells varies according to their geographical situations. The variations are described in a matrix.

The problem is to assign frequencies to transmitters to avoid interference and, if possible, to minimize the number of frequencies.

Code Sample 7.1 shows the OPL model for this example. The model file is `alloc.mod` in your product distribution.

```

int nbCells = ...;
int nbFreqs = ...;
range Cells 1..nbCells;
range Freqs 1..nbFreqs;
int nbTrans[Cells] = ...;
int distance[Cells,Cells] = ...;

struct TransmitterType {Cells c; int t;};
{TransmitterType} Transmitters = {<c,t> | c in Cells & t in 1..nbTrans[c]};
var Freqs freq[Transmitters];
var Freqs maxFreq;

solve {
  forall(c in Cells & ordered t1, t2 in 1..nbTrans[c])
    abs(freq[<c,t1>] - freq[<c,t2>]) >= 16;

  forall(ordered c1, c2 in Cells : distance[c1,c2] > 0)
    forall(t1 in 1..nbTrans[c1] & t2 in 1..nbTrans[c2])
      abs(freq[<c1,t1>] - freq[<c2,t2,>]) >= distance[c1,c2];
};

search {
  forall(t in Transmitters ordered by increasing <dsize(freq[t]),nbTrans[t.c]>)
    tryall(f in Freqs ordered by decreasing nbOccur(f,freq)){
      freq[t] = f;
    };
  maxFreq = sum(i in Freqs) (nbOccur(i,freq) > 0);
};

```

**Code Sample 7.1** OPL Model for the Frequency Allocation Example with Search Procedure

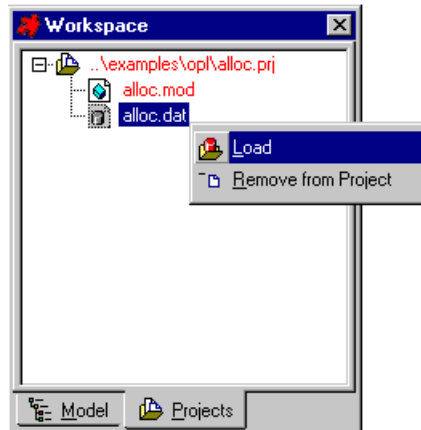
Code Sample 7.2 shows the data initialization for the problem. This code can be found in the `alloc.dat` file of the product distribution.

```
nbCells = 25;
nbFreqs = 256;
nbTrans = [8 6 6 1 4 4 8 8 8 8 8 4 9 8 4 4 10 8 9 8 4 5 4 8 1 1];
distance = [
    [16 1 1 0 0 0 0 0 1 1 1 1 1 2 2 1 1 0 0 0 2 2 1 1 1]
    [1 16 2 0 0 0 0 0 2 2 1 1 1 2 2 1 1 0 0 0 0 0 0 0 0]
    [1 2 16 0 0 0 0 0 2 2 1 1 1 2 2 1 1 0 0 0 0 0 0 0 0]
    [0 0 0 16 2 2 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
    [0 0 0 2 16 2 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
    [0 0 0 2 2 16 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
    [0 0 0 0 0 0 16 2 0 0 1 1 1 0 0 1 1 1 1 2 0 0 0 1 1]
    [0 0 0 0 0 0 2 16 0 0 1 1 1 0 0 1 1 1 1 2 0 0 0 1 1]
    [1 2 2 0 0 0 0 0 16 2 2 2 2 2 2 1 1 1 1 1 1 0 1 1]
    [1 2 2 0 0 0 0 0 2 16 2 2 2 2 2 1 1 1 1 1 1 0 1 1]
    [1 1 1 0 0 0 1 1 2 2 16 2 2 2 2 2 2 1 1 2 1 1 0 1 1]
    [1 1 1 0 0 0 1 1 2 2 2 16 2 2 2 2 2 1 1 2 1 1 0 1 1]
    [1 1 1 0 0 0 1 1 2 2 2 2 16 2 2 2 2 1 1 2 1 1 0 1 1]
    [2 2 2 0 0 0 0 0 2 2 2 2 2 16 2 1 1 1 1 1 1 1 1 1]
    [2 2 2 0 0 0 0 0 2 2 2 2 2 2 16 1 1 1 1 1 1 1 1 1]
    [1 1 1 0 0 0 1 1 1 1 2 2 2 1 1 16 2 2 2 1 2 2 1 2 2]
    [1 1 1 0 0 0 1 1 1 1 2 2 2 1 1 2 16 2 2 1 2 2 1 2 2]
    [0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 16 2 2 1 1 0 2 2]
    [0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 16 2 1 1 0 2 2]
    [0 0 0 1 1 1 2 2 1 1 2 2 2 1 1 1 1 2 2 16 1 1 0 1 1]
    [2 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 1 1 1 16 2 1 2 2]
    [2 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 1 1 1 2 16 1 2 2]
    [1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 16 1 1]
    [1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 16 2]
    [1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 2 16]];
};
```

**Code Sample 7.2** OPL Data for the Frequency Allocation Example (`alloc.dat`)

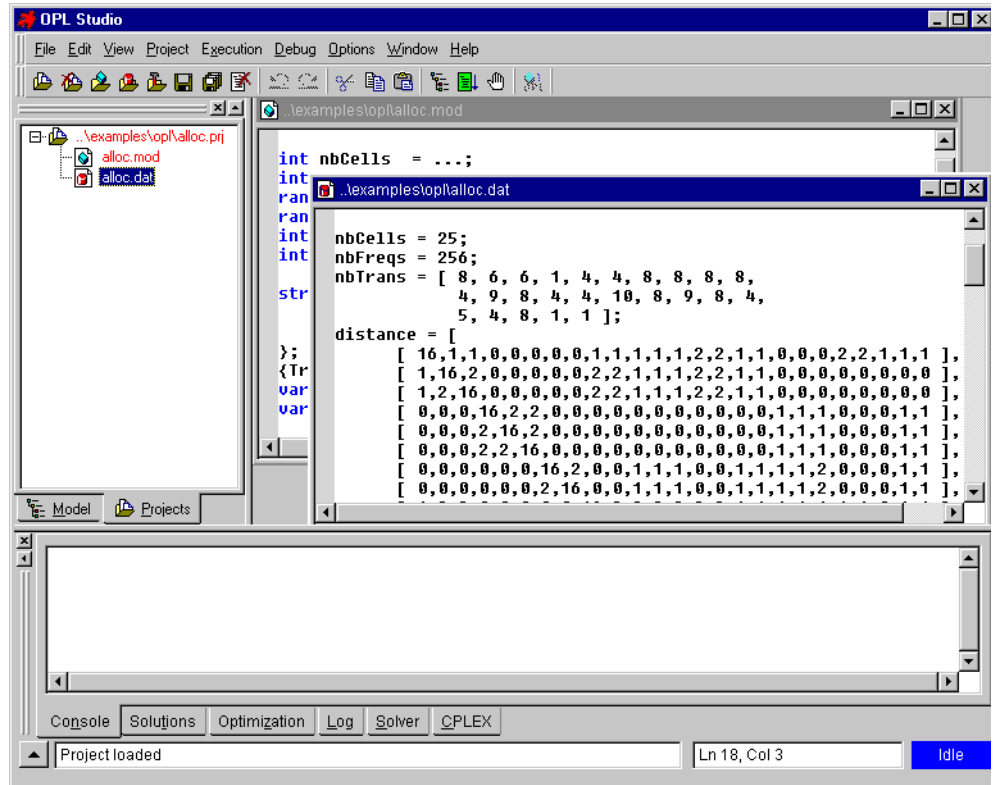
Select **Open>Project** from the File menu, or click on the Load Project button in the tool bar, then select `alloc.prj` from the list and click Open.

The project tree showing the associated model and data files is displayed on the Projects page of the work space. For this example, we will load the data into the editor.



**Figure 7.2** Project Tree for the Frequency Allocation Example (in Dockable Window)

ILOG OPL Studio opens the files associated with the project in the editing area of the Main window. For this example, it opens `alloc.mod` and `alloc.dat`.

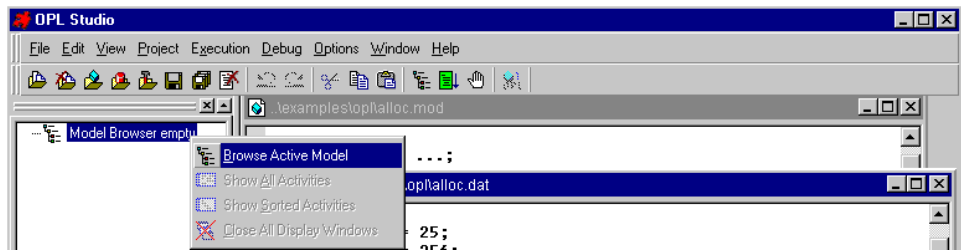


**Figure 7.3** Main Window for the Frequency Allocation Example

## Looking at the Model Structure

If you want to see the data structure of `alloc.mod` before executing the project, you need to browse the model. To do this:

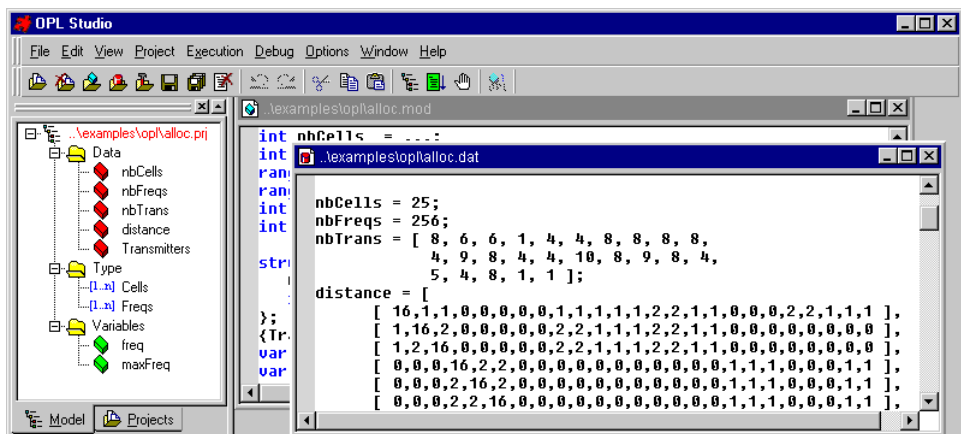
- ◆ click on the Model tab in the work space
- ◆ right-click on the root item
- ◆ select Browse Active Model from the menu (see Figure 7.4).



**Figure 7.4** Browsing an Active Model - Frequency Allocation Example


Other ways of browsing a model are described in *Using the Model Browser* on page 67.

The model's elements are displayed in a tree structure in the work space (see Figure 7.5).



**Figure 7.5** Model Browser for the Frequency Allocation Example

To expand the docked work space window, do one of the following:

- ◆ drag down the border between the work space and the output area and drag to the right the border between the work space and the editing area
- ◆ click on the button  in the top right corner of the work space.

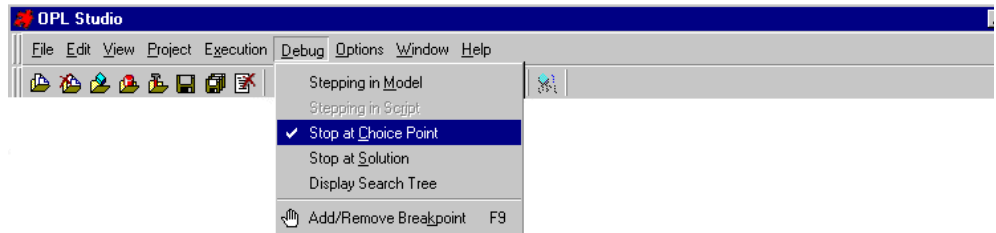
## Setting the Debug Option

The frequency allocation example illustrates the debug feature well, because of the search procedure that appears in the model file. (See Code Sample 7.1.) To use the debug feature, select one of the debug options from the Debug menu in the menu bar.

ILOG OPL Studio gives you the following options for a model debugging session:

- ◆ **Stepping in model** – ILOG OPL Studio stops at each instruction in the search procedure.
- ◆ **Stop at choice point** – ILOG OPL Studio stops at each choice point.
- ◆ **Stop at solution** – ILOG OPL Studio stops at each solution in an optimization.
- ◆ **Display Search Tree** – The search tree is displayed.
- ◆ **Add/Remove Breakpoint** – A breakpoint breaks the execution in the search procedure. This is useful when you want to run quickly through part of the execution, then stop and display the value of a variable, or continue the execution by clicking Next.

To begin this example, we will run ILOG OPL Studio in debug mode and stop at each choice point. Select the option Stop at Choice Point. Once you have selected a debug option from the menu, ILOG OPL Studio is in debug mode.




*Figure 7.6 Debug Menu*



## Executing the Frequency Allocation Project

You can now execute the project with ILOG OPL Studio in debug mode.

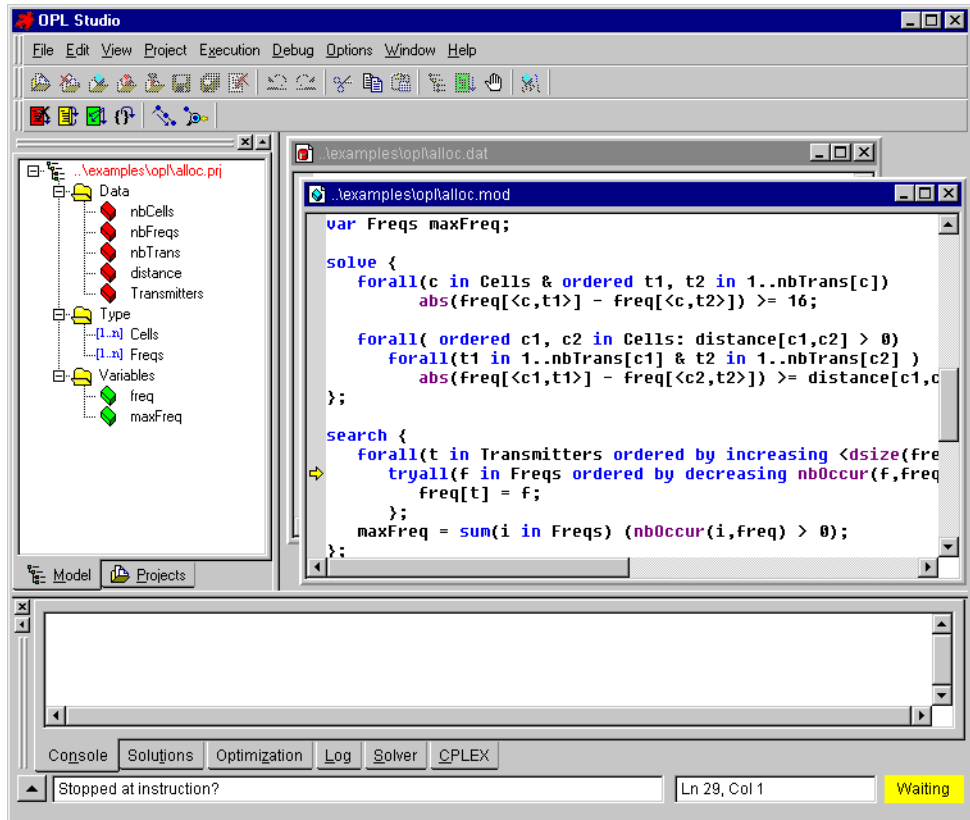
Click the Run button  in the tool bar of the Main window.

**Note:** *If you happen to have other models open in the Main window, ILOG OPL Studio always gives precedence to the model associated with the active project. The active project model always gets executed. To find out which project is the active one, look at the Projects notebook page; the active project is colored red.*

During the execution, ILOG OPL Studio:

- ◆ analyzes the model and, if not already done, produces the summary information and updates the structure in the model browser
- ◆ checks for syntactic or semantic errors
- ◆ executes the model displaying “OPL Studio is running” in the status bar, showing the name `alloc.prj` in the Path Name area, and changing the color patch to green
- ◆ stops at the first choice point
- ◆ enters a waiting state, displays “Stopped at instruction?” in the status bar, displays the line number and column number of the cursor in the editor, and changes the color patch to yellow.

The Main window now looks like this:



**Figure 7.7** Main Window in Debug Mode with Stop at Choice Point Option

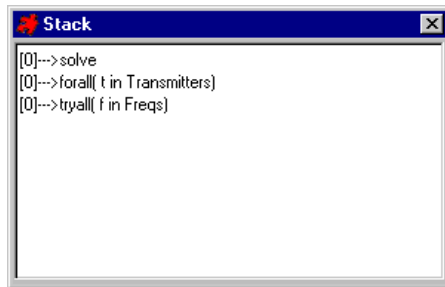
The yellow arrow in the editor margin indicates the instruction at which execution stops. Now you can display two additional windows that will help you perform your debugging task:

- ◆ the Stack window
- ◆ the Inspector window.

## Displaying the Stack Window

Click the View Choice Stack button  in the tool bar of the Main window.

ILOG OPL Studio displays the Stack window in a docked state, but you can float it by clicking on its handle at the left side. All the instructions executed so far appear in the Stack window. After stopping at the first choice point, the window looks like this:

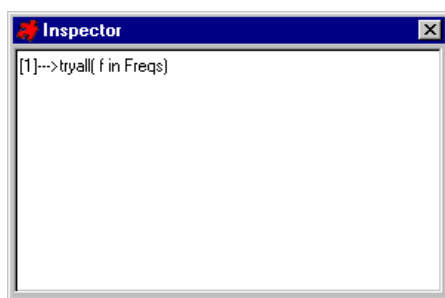


**Figure 7.8** Stack Window (Floating) at the First Choice Point

## Displaying the Inspector Window

Click the Inspect Current Node button  in the tool bar of the Main window.

ILOG OPL Studio displays the Inspector window in a docked state, but you can float it by clicking on its handle at the left side. The current instruction is displayed in the Inspector window.



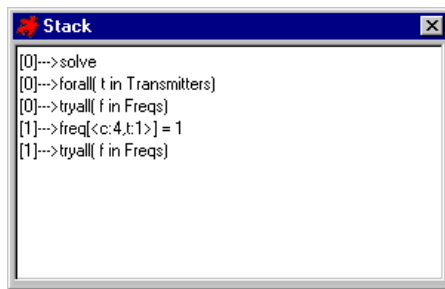
**Figure 7.9** Inspector Window (Floating) at the First Choice Point

## Continuing the Execution

You can continue the execution by clicking the Next button to look at the subsequent choice points.

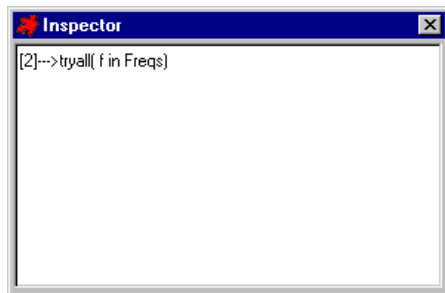
Click the Next button  to go to the next choice point.

As you do this, you can see how the Stack and Inspector windows change, showing the instructions that were executed. The Stack window now looks like this:



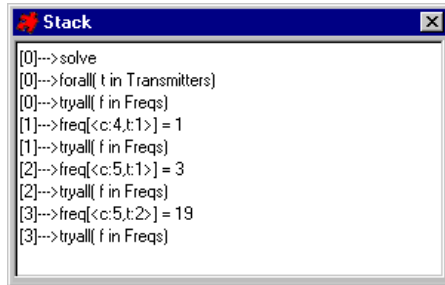
**Figure 7.10** Stack Window at Second Choice Point

The Inspector window now looks like this:

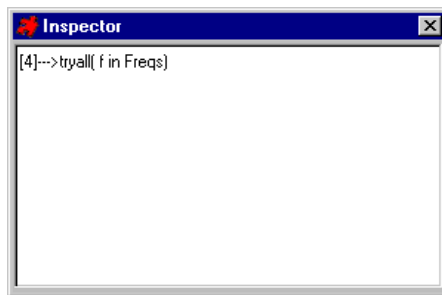


**Figure 7.11** Inspector Window at Second Choice Point

Click the Next button two more times. As you progress through the execution, you can see the values that were assigned to the variables. The Stack and Inspector windows should now look like this:




**Figure 7.12** Stack Window at the Fourth Choice Point




**Figure 7.13** Inspector Window at the Fourth Choice Point

Now, you can do one of the following:

- ◆ If you want to continue, step by step, looking at the values assigned to the variables, click the Next button until you reach the end of the execution.
- ◆ If you want to continue the execution without stopping, click the Continue Run button  in the tool bar.

When you click the Continue Run button at this point, ILOG OPL Studio leaves debug mode and finds the first solution. This solution appears in the Solutions notebook page. ILOG OPL Studio then enters its waiting state so that you can ask for the next solution or complete the execution.

- ◆ If you have checked what you are looking for, you can terminate the execution by clicking the Abort button  in the execution tool bar.

For this example, we will terminate the execution so that you can look at another way of using the debug feature.

Click the Abort button. ILOG OPL Studio returns to its idle state. Notice that the model browser collapses, and the Stack and Inspector windows are closed.


Next, you are going to see how the Stepping option of the debug feature works.

---

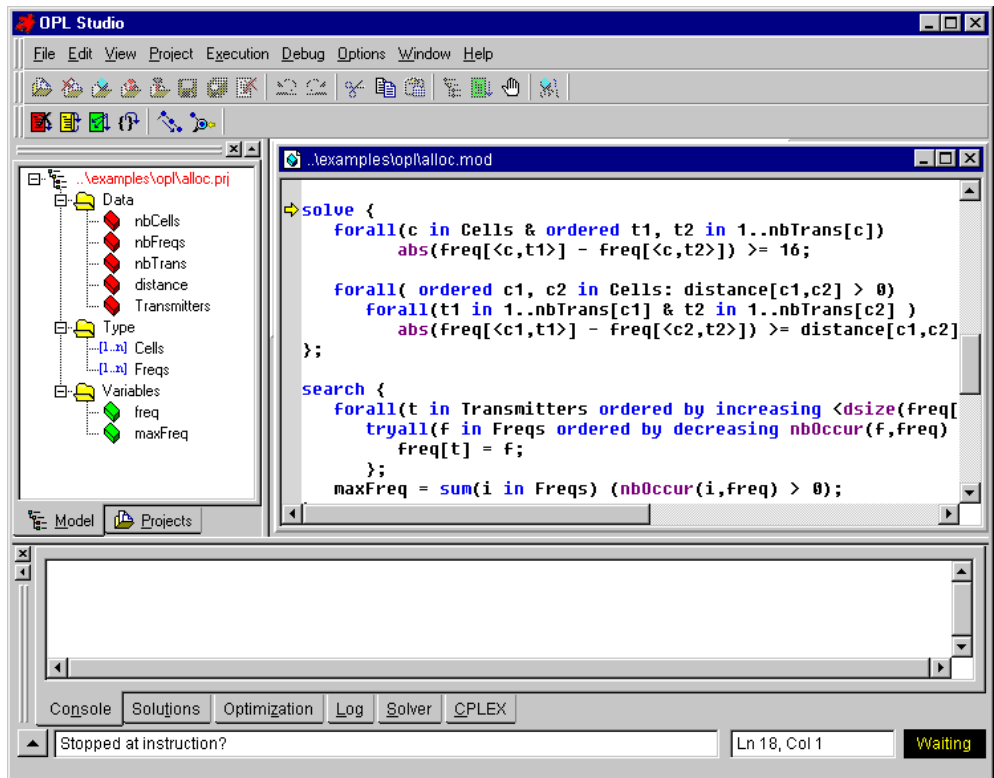
## Executing the Project with the 'Stepping in Model' Option

The Stepping in Model option from the Debug menu allows you to follow the execution of the model more closely by looking at each instruction. The difference between the Stepping in Model option and the Stop at Choice point option is as follows. When you use Stop at Choice Point, you see only the values that were actually assigned during the execution. When you use Stepping in Model, you see all the values that ILOG OPL Studio tries to assign during the execution, including the ones leading to failures. With the Stepping in Model option, you can see a more detailed view of the execution.

Go to the Debug menu and deselect the Stop at Choice Point option, then select the Stepping in Model option.

Now, click the Run button  to execute ILOG OPL Studio in debug mode.

When ILOG OPL Studio stops at the first instruction, the Main window looks like this:



7. Debugging the Search Strategy

**Figure 7.14** Main Window in Debug Mode with Stepping Option

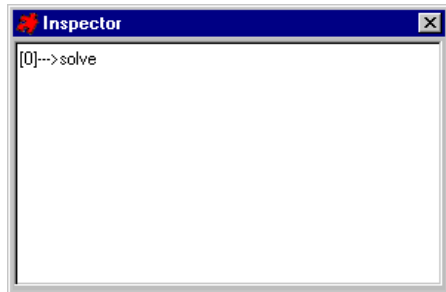
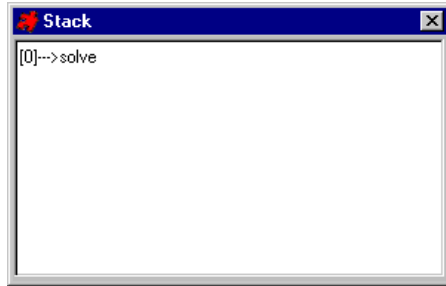
The first instruction is indicated by the yellow current line arrow in the editor.

---

### Displaying the Stack Window and Inspector Window

Now, open the Stack and Inspector windows. Click those buttons in the tool bar.

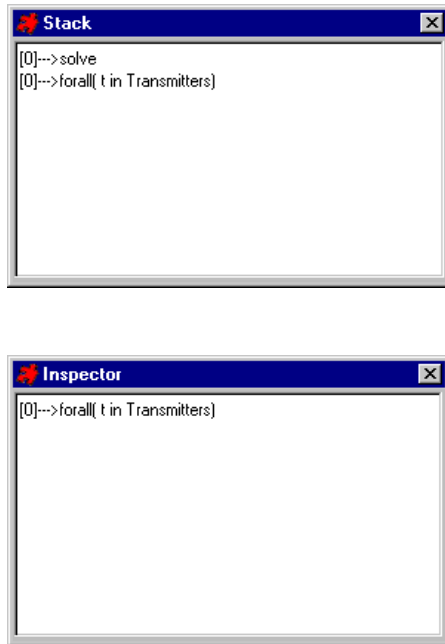
The Stack window will again display all the instructions executed in the program. The Inspector window will again display the current instruction executed.



*Figure 7.15 Stack and Inspector Windows at First Instruction of Stepping Option*



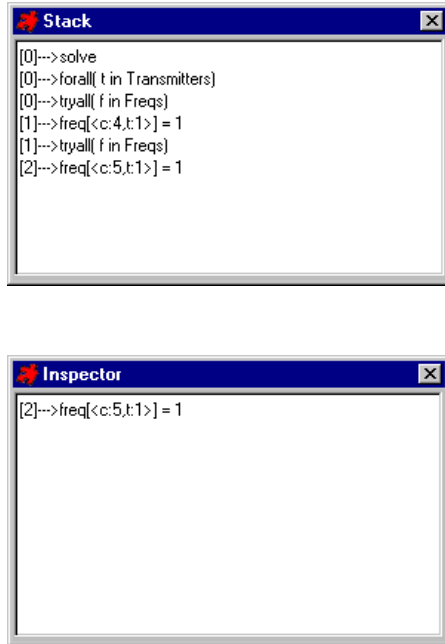
Click the Next button to move to the next instruction. You can see how the windows have changed.



**Figure 7.16** *Stack and Inspector Windows at Second Instruction of Stepping Option*

Click the Next button four more times. While you are doing this, notice how ILOG OPL Studio moves from instruction to instruction and how the Stack and Inspector windows change with each move.

The Stack and Inspector windows now look like this:



**Figure 7.17** *Stack and Inspector Windows at Sixth Instruction of Stepping Option*

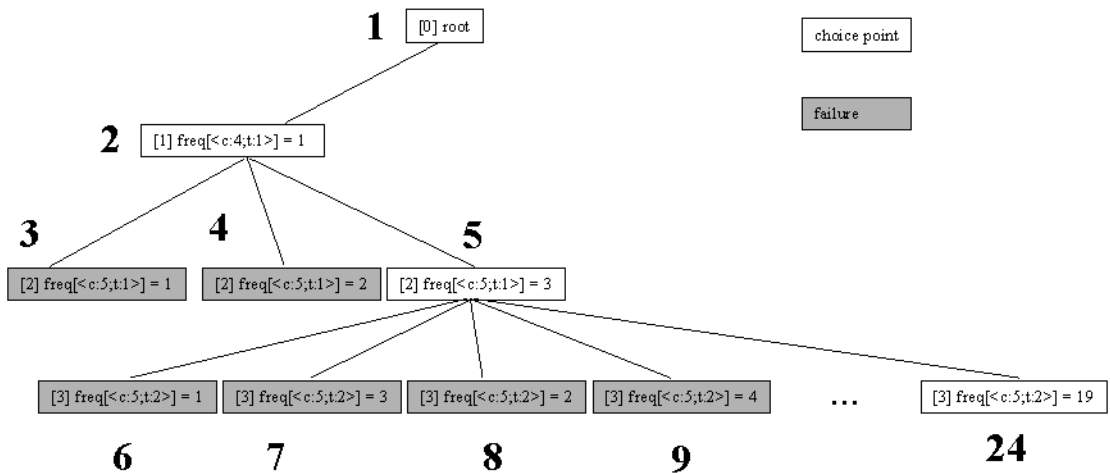
If you click the Next button several more times, you will see how ILOG OPL Studio displays the values that it tries to assign to the variables.

Because the values 1 and 2 assigned to `freq[<c:5;t:1>]` lead to a failure, you did not see them with the option Stop at Choice Point. Here, with the option Stepping in Model, you can also see those attempts.

## Visualizing the Search Strategy with the Stack and Inspector Windows

The Stack window displays the current path inside the search tree.

To understand more clearly what is displayed in the Stack and Inspector windows during a search, we will examine the corresponding search tree.



**Figure 7.18** Order of the Search Tree Exploration with the Default Depth-First Search

In this search tree, the path 1-2-5-24 corresponds to the following choices:

- ◆ the value 1 has been assigned to the variable `freq[<c:4;t:1>]` and
- ◆ the value 3 has been assigned to the variable `freq[<c:5;t:1>]` and
- ◆ the value 19 has been assigned to the variable `freq[<c:5;t:2>]`.

When executing with the Stop at Choice Point debug option, only the choice point nodes are displayed in the Stack and Inspector Windows; the failure nodes are hidden. When executing with the Stepping in Model debug option, you see all the nodes, including the choices that have failed. The number between brackets indicates the depth in the search tree.

At the same level, the same variable is selected but different values are tried. Notice that the order in which the values must be tried is specified by the statement:

```
ordered by decreasing nbOccur(f,freq)
```

For instance, look at the 7<sup>th</sup> and 8<sup>th</sup> nodes. You will notice that the value 3 is tried before the value 2 for the variable `freq[<c:5;t:2>]`. In the path 1-2-5, the frequencies 1 and 3 are used once each for the variables `freq[<c:4;t:1>]` and `freq[<c:5;t:2>]` respectively. The value 2 is not yet used, so has lower priority than the values 1 and 3. You see that the statement prioritizes the use of already allocated frequencies.

Click on the Abort button to stop execution of the project.

## Visualizing the Search Tree

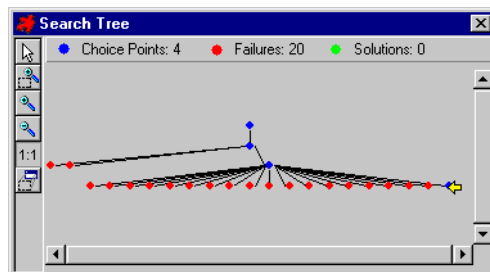
We are going to execute `alloc.prj` again, this time displaying the search strategy in the search tree. Two search procedures will be used:

- ◆ the Depth First Search
- ◆ the Slice-Based Search.

### Using the Depth First Search

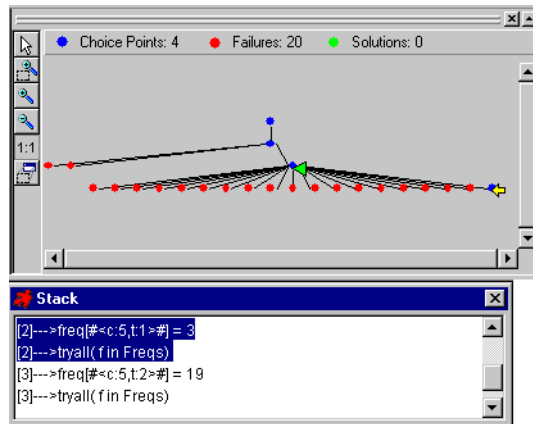
When declaring a search specification, you declare the search tree that the OPL engine must explore. You do not specify how the search tree is to be explored. By default, the search tree is explored by a Depth First Search.

1. In the Debug Menu, deselect Stepping in Model, then select the Display Search Tree option in addition to Stop at Choice Point.
2. Execute the project by clicking on Run.
3. Select the View>Choice Stack menu item.
4. Click on Next 3 times. Note that, with the option Stop at Choice Point, OPL Studio does not stop at the fail nodes. The n-ary tree corresponding to your search is displayed.



The yellow arrow indicates the current node.

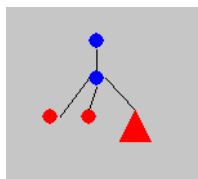
5. Double-click on the second, and then on the third node from the root.



A green arrow appears next to the node you have selected by the double-click and the corresponding lines in the goal stack are highlighted.

**Note:** You must select a node in the active branch, that is, a node in the path from the current node (yellow arrow) to the root.

6. Right click on the third node. In the context-sensitive menu that appears, select the Shrink option. The subtree is shrunk and replaced by a triangle, the color of which depends on the nodes that have been shrunk. If a solution has been found in the subtree, the triangle is green. If the subtree contains only blue nodes (Interior nodes, i.e. choice points), then it is blue. If there are failures in the subtree and no solution found, it is red.



## Using the Slice-Based Search

You can select other search strategies, including the Slice-Based Search (SBS). See the *ILOG OPL Studio: Language Manual* for more information on search strategies. We will now try the SBS strategy to see what the Stack and Inspector windows display.

1. Stop the execution with the Abort button.
2. Select the SBS as follows:
  - open the Project Options dialog box via the command:  
Options>Customize Active Project Options
  - in the Constraint Programming page, select Slice-Based Search from the Procedure combo-box
  - Click on OK.
3. Now rerun the project with the Stepping in Model debug option. (Don't forget to deselect the previous options.)
4. Click on the buttons View Choice Stack and Inspect Current Node to make the corresponding docking windows appear.
5. Click on the Next button, 10 times.

The Stack window should display:

```
[0]--->solve
[0]--->forall(t in Transmitters)
[0]--->tryall(f in Freqs)
[1]--->freq[#<c:4,t:1>#] = 1
[1]--->tryall(f in Freqs)
[2]--->freq[#<c:5,t:1>#] = 3
[2]--->tryall(f in Freqs)
[3]--->freq[#<c:5,t:2>#] = 3
```

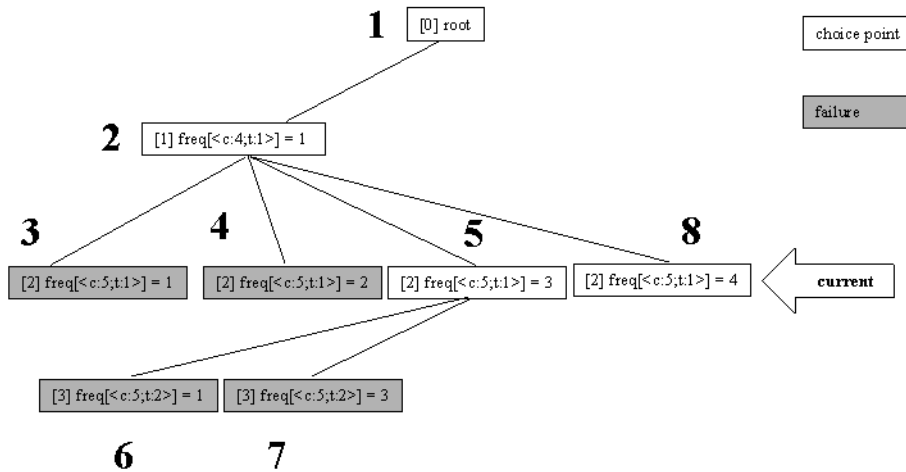
6. Click on the Next button an 11<sup>th</sup> time.

The Stack window should now display:

```
[0]--->solve
[0]--->forall(t in Transmitters)
[0]--->tryall(f in Freqs)
[1]--->freq[#<c:4,t:1>#] = 1
[1]--->tryall(f in Freqs)
[2]--->freq[#<c:5,t:1>#] = 4
```

It has jumped in the tree.

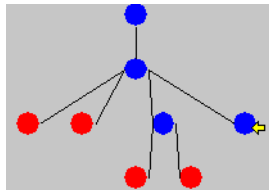
The corresponding search tree exploration order is as follows:



**Figure 7.19** Part of the Search-tree Exploration Order with the SBS

You see that the SBS explores the same tree in a different order. See the *ILOG OPL Studio: Language Manual* for more information.

You can abort and rerun with the Display Search Tree item checked. After clicking 11 times on the Next button, you will obtain the following tree layout:



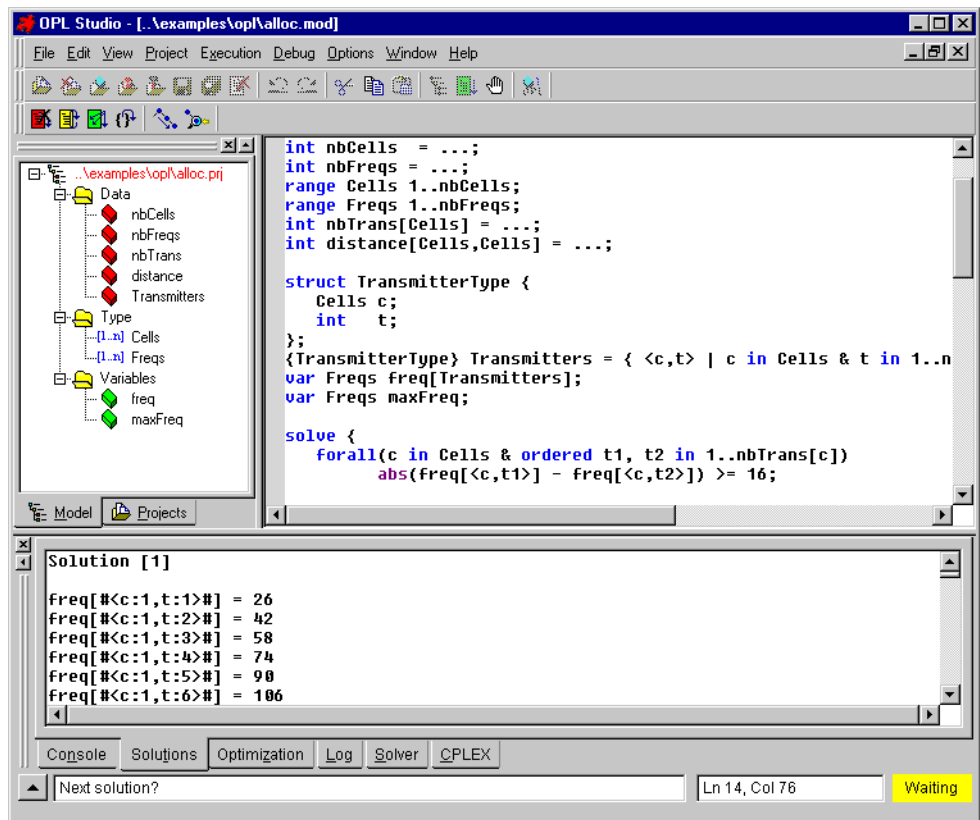
When you have a feel for how the Stepping in Model option works, stop the execution and reopen the Project Options dialog box. Return to the Depth First Search procedure which is, in fact, more suitable in this example. Uncheck the Debug options and run the project again.

Note that an example demonstrating the usefulness of the SBS strategy is given at the following location:

`examples\opl\scheduler\jobshop20.prj`


To compute the optimal solution and prove the optimality, this model takes days using Depth First Search and seconds with Slice-Based Search.

The Main window now looks like this:



**Figure 7.20** Main Window After Leaving Debug Mode

ILOG OPL Studio displays the first solution and enters the waiting state, expecting further direction on how to proceed.

- ◆ You can look at further solutions by clicking the Next button or the Continue Run button. ILOG OPL Studio continues the execution and produces all the remaining solutions, if they exist.
- ◆ Otherwise, to terminate the execution, you can click the Abort button .



## Cooperating Solvers – Combined LP and CP

The keywords `minimize with linear relaxation`, instead of `minimize`, ensure that OPL uses the linear relaxation on all linear constraints. Thus CPLEX will produce a lower bound at each node of the Solver search tree. In this section we will take the example `wareboth.prj` to visualize the search tree, which will be explored in both cases:

- ◆ using linear relaxation
- ◆ not using linear relaxation.

First, open the project:

```
examples\opl\wareboth.prj
```

The example is described in the *ILOG OPL Studio: Language Manual*. In this problem, a company is considering a number of locations for building warehouses to supply its existing stores. Each possible warehouse has a fixed maintenance cost and a maximum capacity specifying how many stores it can support. In addition, each store can be supplied by only one warehouse and the supply cost to the store varies according to the warehouse selected. The application consists of choosing a) the warehouses to build and b) which of them should supply the various stores in order to minimize the total cost, i.e., the sum of the fixed and supply costs.

This model does not contain only linear constraints over integer expressions. The model contains higher order constraints. See the constraint:

```
forall(w in Warehouses)
    sum(s in Stores) (supplier[s] = w) <= capacity[w];
```

`supplier[s] = w` contains a variable, so it is a constraint, not a traditional expression. Therefore the overall constraint is a meta-constraint (a constraint on constraints).

If we do not use explicitly `with linear relaxation`, OPL switches to Solver alone.

1. First you need to modify the example. Put the objective function in a variable so that you can refer to it graphically. Create `totalCost`, an integer variable between 0 and 1000.

```
.....
var int totalCost in 0..1000;
```

Now minimize `totalCost` and add a constraint to link this variable to the objective function expression.

```
minimize with linear relaxation
    totalCost
subject to {
    totalCost = sum(w in Warehouses) fixed * open[w] +
                sum(w in Warehouses, s in Stores)
                supplyCost[s,w] * supply[s,w];
    .....
};

search {
    forall(s in Stores ordered by decreasing regretdmin(cost[s]))
        tryall(w in Warehouses ordered by increasing supplyCost[s,w])
            supplier[s] = w;
};
```

2. Select Display Search Tree from the Debug menu. (Deselect any other options.)
3. Run the program. You will see a small search tree, with only a few branches. The optimal value in the Solutions panel is 383. Click on Abort to stop the search.
4. Comment the keywords with `linear relaxation`.

```
minimize // with linear relaxation
    totalCost
```

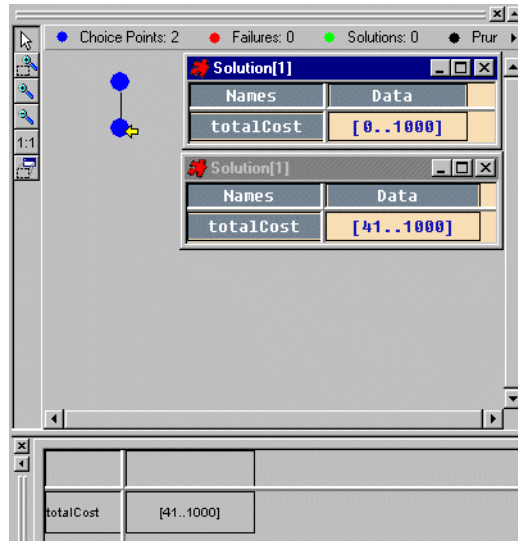
5. Run the program again. This time the tree is much bigger. Click on Abort.
6. In the Debug menu, select Stop At Choice Point and Stop At Solution.
7. In the Execution menu, select Browse Active Model.
8. In the Model Browser Notebook Page, right click on the `totalCost` variable and select the option “display domain”.
9. Run the program again. Select View>Choice Stack or click on the corresponding item on the execution tool bar.

The root of the tree appears. The domain of the objective is 0..1000.

Double-click on the `totalCost` item inside the model browser. You can keep a snapshot of the domain’s current state.

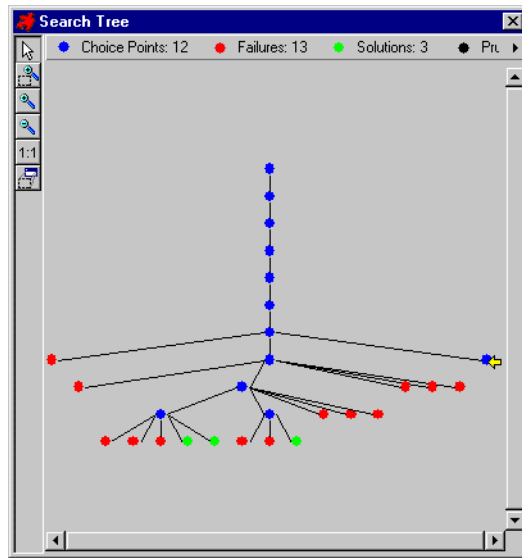
10. Click once on Next. You see the domain of the objective reduced to 41..1000 in the lower notebook.

Double-click on the `totalCost` item inside the model browser. You can keep another snapshot of the domain's current state.



**Figure 7.21** Search Tree and Objective Bounds, after One Choice using Solver without Linear Relaxation

11. Now click on the Next button six more times. You will see OPL finding solutions and backtracking.



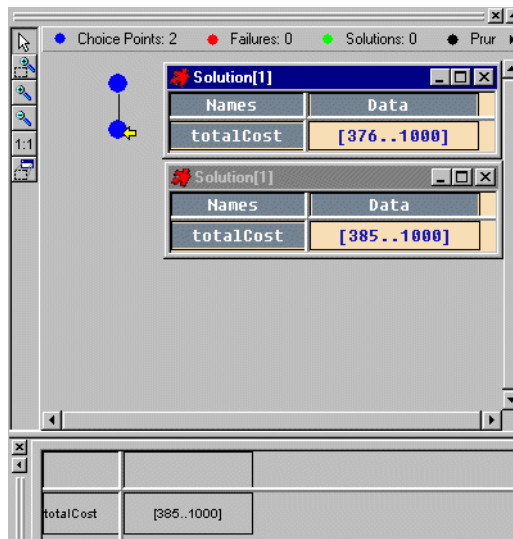
**Figure 7.22** Search Tree without Linear Relaxation after 14 Next Commands

12. Click on the Abort button.

Uncomment the keywords with `linear relaxation`. Run the program again.

The root of the tree appears. The domain of the objective is 376..1000. CPLEX produced a first lower bound: 376.

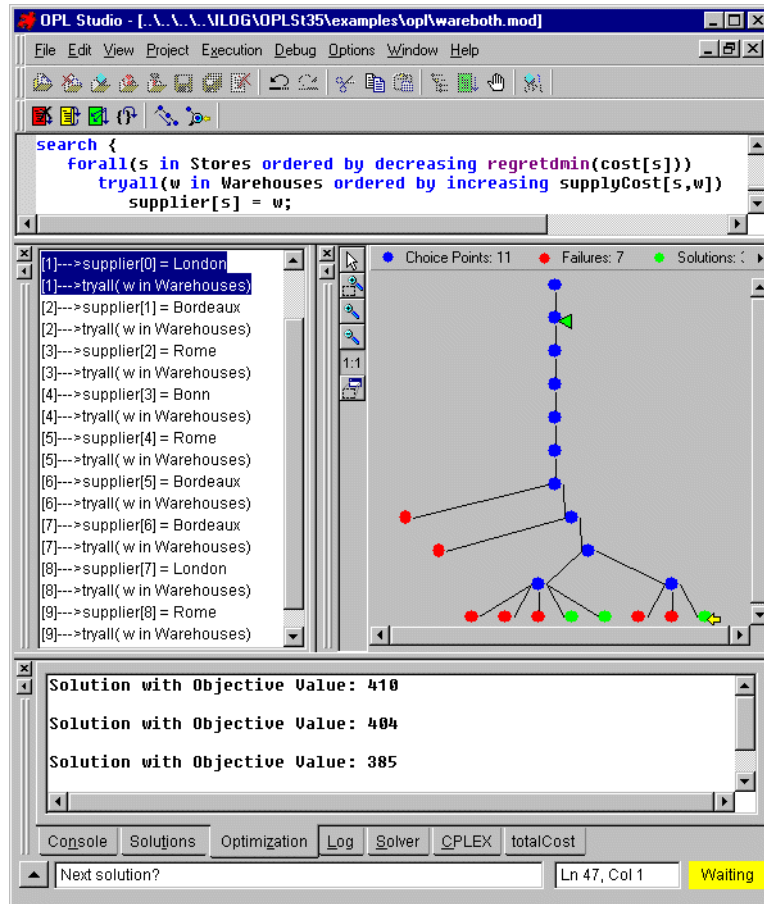
13. Double-click on the `totalCost` item inside the model browser. You can keep a snapshot of the domain's current state.
14. Click one time on Next. This second node represents the first choice. You see the domain of the objective reduced to 385..1000 in the lower notebook. CPLEX produced a lower bound at 385. Double-click on the `totalCost` item inside the model browser. You can keep another snapshot of the domain's current state.



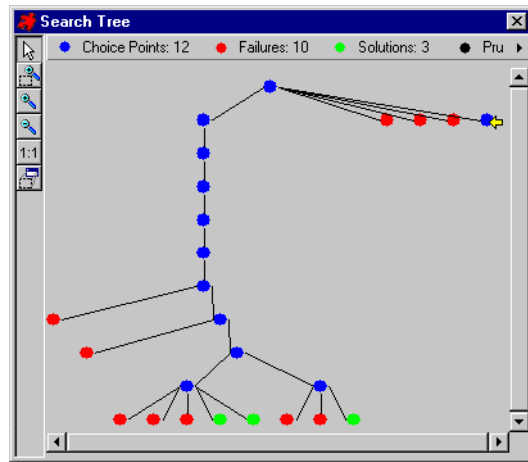
**Figure 7.23** Search Tree after One Choice using Solver with Linear Relaxation at Each Node

15. Click 12 more times on the Next button. You see OPL finding 3 solutions. The third one has 385 as objective value, that is the lower bound of the `totalCost` domain as computed by the linear relaxation. Double click on the first choice, i.e. the child of the root node. The Choice Stack window appears and the label corresponding to the selected node is highlighted. The first choice sets the variable `supplier[0]` to the value London.

You can drag and drop to see the complete choice stack together with the search tree:



16. Click one more time on the Next button. This time the search explores fewer nodes and backtracks sooner to the root children, pruning the other potential nodes. The pruned nodes are not displayed. Compare this to the previous tree obtained without the linear relaxation. This time, because you know that the lower bound of the objective value is 385 in the subtree beginning at the root's first child, it is unnecessary to search for a better solution than the third solution found (Objective Value: 385). So Solver prunes this subtree and explores the brothers of the root's first child.



**Figure 7.24** Search Tree with Linear Relaxation after 14 Next commands

---

## Exploration Strategy – Drawing Board Combined with Search Tree

In this section you will visualize simultaneously the search tree and a 2D graphic representation of the choices made by the algorithm at each search node.

1. Open the file:

`examples\opl\eulergr.mod`

2. Select the following options from the Debug menu:

Display Search Tree

Stop At Choice Point

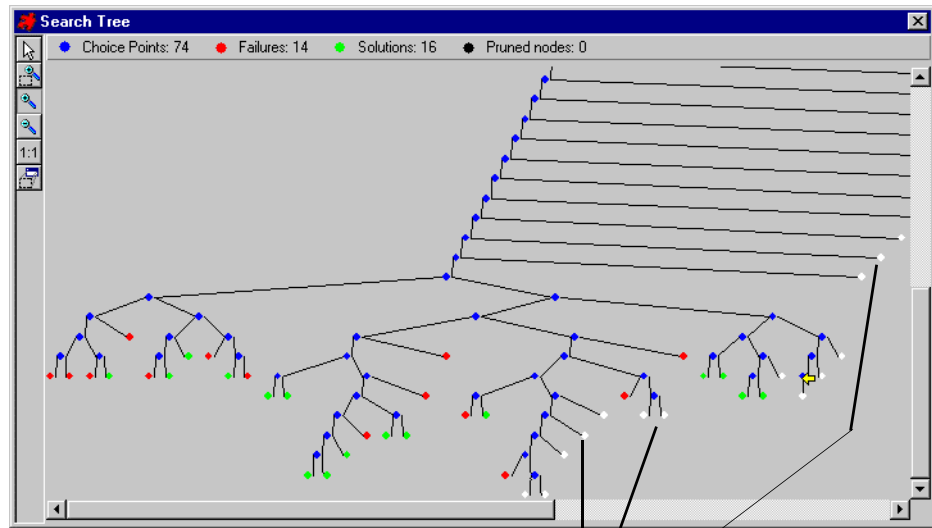
Stop At Solution.

3. Click on the Run button, and click on the Next button several times.

You can visualize the selected knight moves on the chessboard, created simultaneously in the Drawing Board panel and the explored Search Tree. First, notice that you have access to a binary tree. The basic one produced by ILOG Solver. The n-ary tree is available only if you define the choice points yourself with the `try` or `tryall` instructions in the search procedure. To understand what happens at each choice point, take a look at the Drawing Board.

4. Click on the Continue Run button and observe the yellow arrow exploring the tree in a Depth First Search strategy. The white nodes are the nodes created by ILOG Solver and not yet visited.
5. Click on the Abort button. Select Customize Default Options from the Options Menu. Select the Constraint Programming page and select the Slice-Based Search as the search strategy.
6. Deselect Stop At Choice Point and Stop At Solution. Run the example again. This time you see the yellow arrow jumping in the tree and leaving some white nodes behind it. These white nodes are left unvisited for the moment. They will be visited or pruned later.

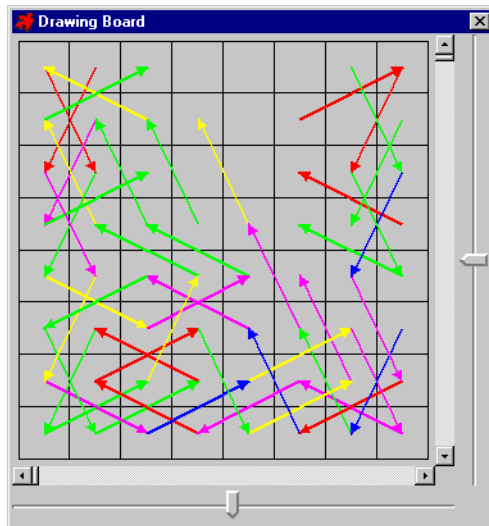




The white dots represent unexplored nodes.

**Figure 7.25** Exploring the Search Tree in the Slice-Based Search

The corresponding state of the Drawing Board is shown below (the colors have no special meaning):



7. Click on the Abort button. Return to the Options dialog box and reselect the Depth First Search as search strategy. Click on OK.

---

## Terminating ILOG OPL Studio

This completes the explanation of displaying the search with ILOG OPL Studio.

This also concludes the ILOG OPL Studio tutorial. By now, you should be able to perform all the basic tasks and start creating and executing your own projects with ILOG OPL Studio.

If you are finished using ILOG OPL Studio, you can terminate the session by selecting Quit from the File menu.

## Customizing ILOG OPL Studio

To customize ILOG OPL Studio to suit your particular needs you can:

- ◆ set constraint programming options
- ◆ specify editor options
- ◆ specify output options
- ◆ specify options for makefile generation
- ◆ indicate paths for model and script files
- ◆ set options for mathematical programming (described in Chapter 9, *Mathematical Programming*).

## Default Options and Project Options

There are two types of options:

- ◆ those that are set for OPL Studio and become the default settings and apply to stand-alone models and scripts
- ◆ those that are set for an active project, and apply only to that project.

**Note:** *The project settings take precedence over the default settings.*

The Default Options and Project Options dialog boxes contain the same options. Only the title bars differ. This chapter describes how to set default options.

### Setting the Default Options

You can customize your ILOG OPL Studio sessions using the Default Options dialog box. (See Figure 8.1.) To access this dialog box, select Options>Customize Default Options in the menu bar.

To set your options for the current session, click Apply or OK.

To reset the default values, click Reset.

Default options are saved in the `default.prj` file, in your home directory on UNIX and in your profile directory on Windows, when you quit the application or after the command Save All. The options will then be restored the next time you start OPL Studio. Only an option whose value differs from its default value is saved in the `default.prj` file.

Options for stand-alone model and script files are set in the Default Options dialog box.

### Setting Project Options

For an active project, there are two ways of accessing the project options:

- ◆ In the menu bar, select Options>Customize Active Project Options
- ◆ In the project tree, right-click on the project name, then select Project Options from the menu.

Options and files associated with the active project are saved by right-clicking on the project name, in the project tree, then selecting Save the Project.

See *Setting Project Options* on page 60.

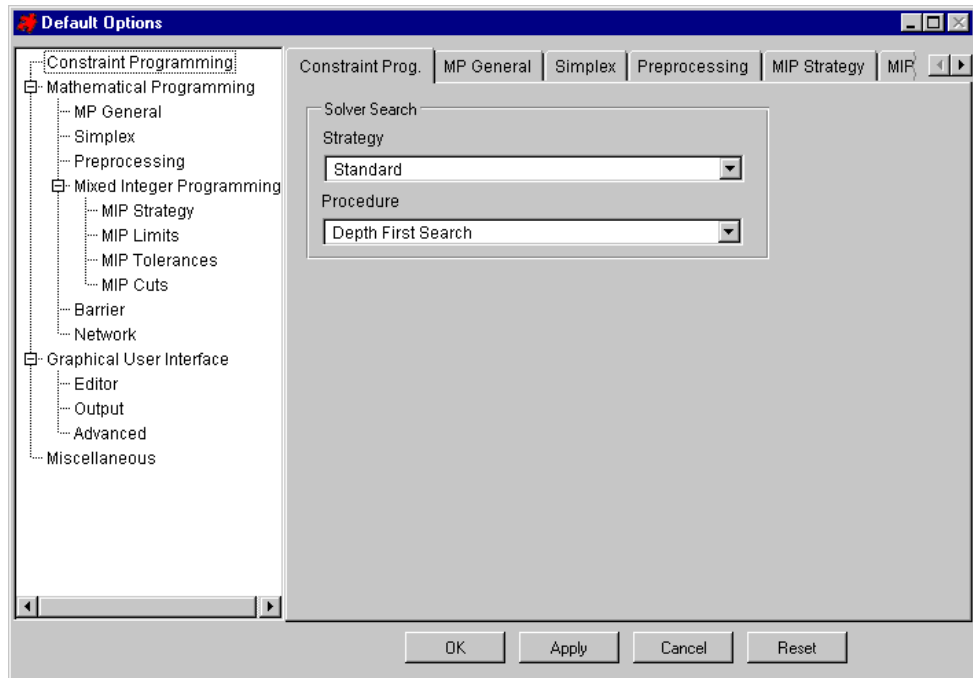
### Navigating in the Options Dialog Boxes

To select a notebook page, either click on a tree item or click on the corresponding tab.

You can also use the navigation arrows in the top right corner.

## Setting Constraint Programming Options

Use the Constraint Prog. notebook page to set the default search strategy and search procedure for ILOG Solver.



**Figure 8.1** Default Options - Constraint Programming

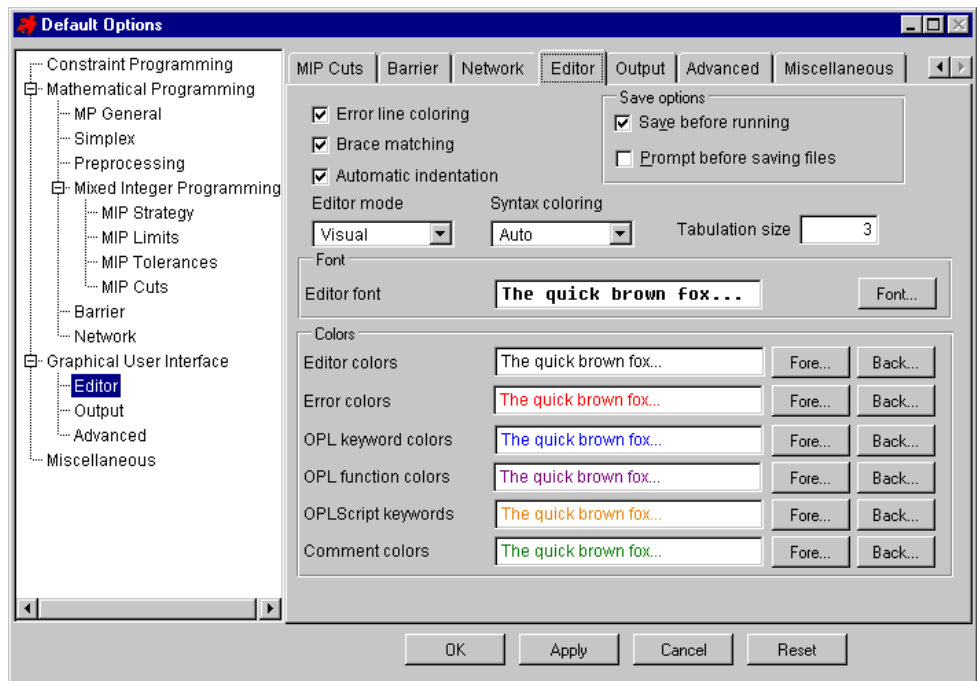
- ◆ **Strategy** – You can choose from:
  - Standard (default)
  - Dichotomic
- ◆ **Procedure** – You can choose from:
  - Depth First Search (default)
  - Sliced-Based Search
  - Depth-bounded Discrepancy Search
  - Best First Search
  - Interleaved Depth First Search

For information on these search procedures, refer to the *ILOG Solver User's Manual*.

## Setting Editor Options

Some of the editor options are described in *The Text Editor* on page 41.

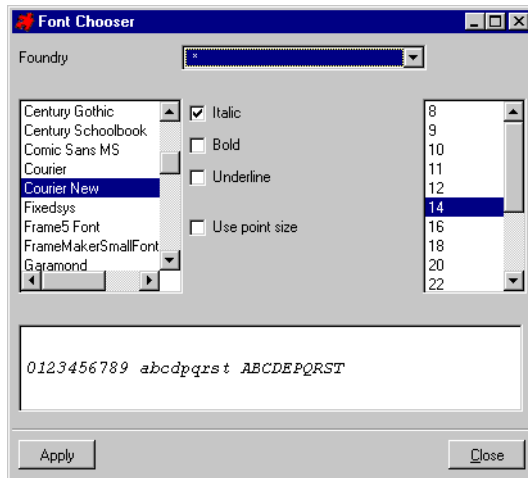
You can use the Editor notebook page to set the default fonts and colors for the ILOG OPL Studio editor.



**Figure 8.2** Editor Options

## Changing the Fonts

You can change the font used for text displays. When you click the Font button, ILOG OPL Studio displays the Font Chooser dialog box.



**Figure 8.3** Font Chooser Dialog Box

Use the list box on the left side of the dialog box to choose the type of font.

Use the check boxes in the middle of the dialog box to choose the style of font (italic, bold, underlined), or the font size in points.

Use the list box on the right side of the screen to specify the size of the font.

The text box at the bottom of the dialog box shows the results of your selection.

Click Apply to return to the Default Options dialog box. You must also click Apply in the Default Options dialog box for your changes to appear in the Main window.

The options will then be restored the next time you start OPL Studio.

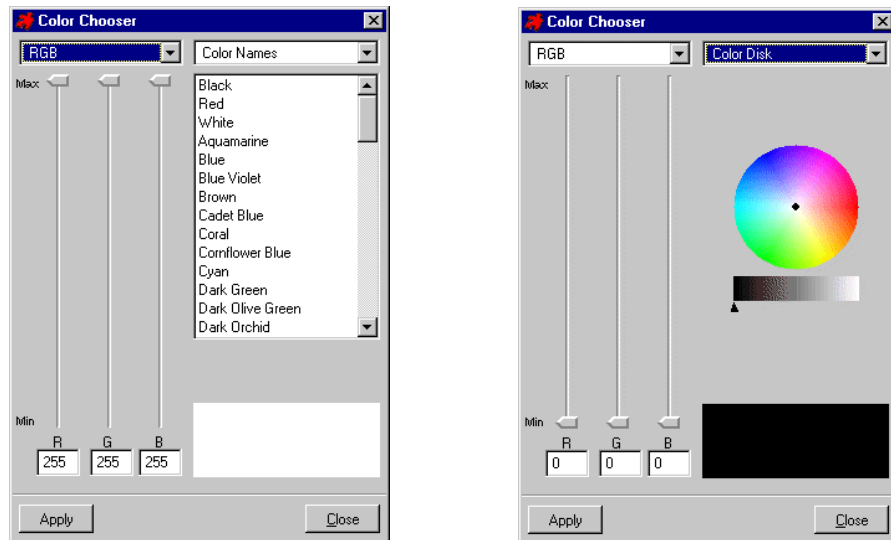
**Tip:** Choose fixed fonts if you have them. They will improve indentation and scrolling performance.

## Changing the Foreground and Background Colors

You can change the color used for the following text displays:

- ◆ **Editor** – the default colors used for ordinary text inside the editor.
- ◆ **Errors** – the colors used to display errors found during syntax checking.
- ◆ **OPL and OPLScript** – the colors used to highlight:
  - OPL and C++ keywords (default: blue and white)
  - OPL and OPLScript functions or methods (default: violet and white)
  - OPLScript keywords and C++ preprocessor macros (default: orange and white)
- ◆ **Comments** – the colors used to display comments in the files you create.

Click the Fore or Back button next to the type of text whose color you want to change. ILOG OPL Studio displays the Color Chooser dialog box. Select Color Names or the Color Disk.



**Figure 8.4** Color Chooser Dialog Boxes

You can use the sliders on the left side of the panels to obtain a color, click the color name in the Names dialog box, or mix colors in the Color Disk.

The color box at the bottom of the dialog box shows the results of your selection.

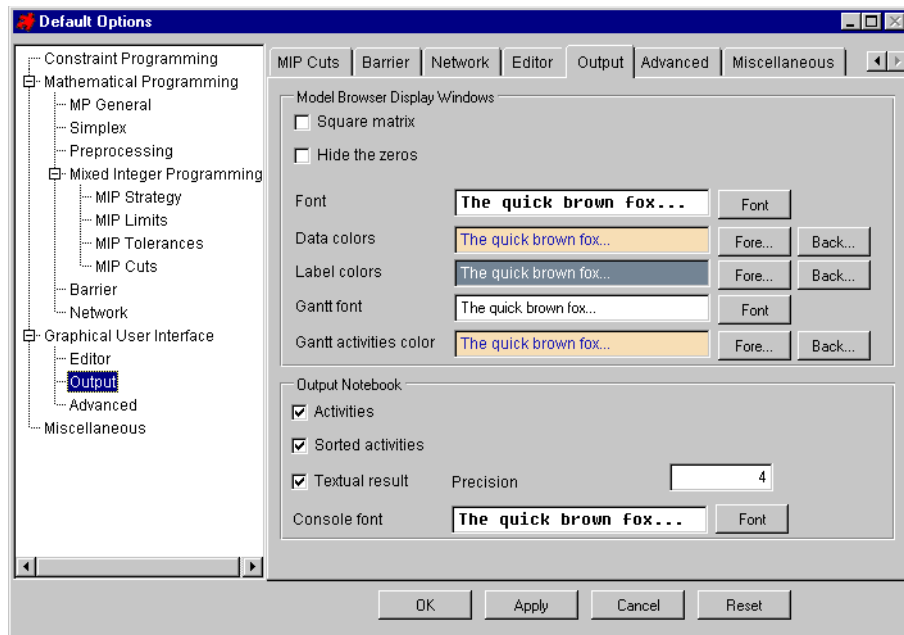
Click Apply to return to the Default Options dialog box. You must also click Apply in the Default Options dialog box for your changes to appear in the Main window.

The options will then be restored the next time you start OPL Studio.



## Setting Output Options

Use the Output notebook page to set the display options for ILOG OPL Studio.



**Figure 8.5** Output Options

- ◆ For the model browser display windows:
  - **Square matrix** – check this box to specify that the cells in a displayed matrix appear as squares. If this box is not checked, the cells appear as rectangles.
  - **Hide the zeros** – check this box to specify that the zeros in a displayed matrix do not appear. If this box is not checked, the zeros appear in the matrix.
  - **Font** – the font used in the results panels displayed from the model browser. Select options from the Font Chooser.
  - **Data colors** – the colors used to display the results in the panels that pop up from the model browser. Select options from the Color Chooser.
  - **Label colors** – the colors used to display the labels of a matrix in the panels that pop up from the model browser. Select options from the Color Chooser.
  - **Gantt font** – the fonts used in the Gantt charts displayed from the model browser.
  - **Gantt activities color** – the activities colors in the Gantt charts displayed from the model browser.

- ◆ For the output notebook:
  - **Activities** – whether or not the Activities notebook page is displayed during the execution of a model with activities defined.
  - **Sorted Activities** – whether or not the Sorted Activities notebook page is displayed during the execution of a model with activities defined.
  - **Textual result** – whether or not ILOG OPL Studio displays the instantiation of the variables in the Solutions and Optimization notebook pages.
  - **Precision** – the number of digits you want to see after the decimal point when displaying a floating point value. The default is 4.
  - **Console Font** – the font used in the Console, Solutions, Optimization, Log and Output notebook pages.

## Setting Advanced Options

Use the Advanced notebook page to set colors for activity domains and docking options for the drawing board.

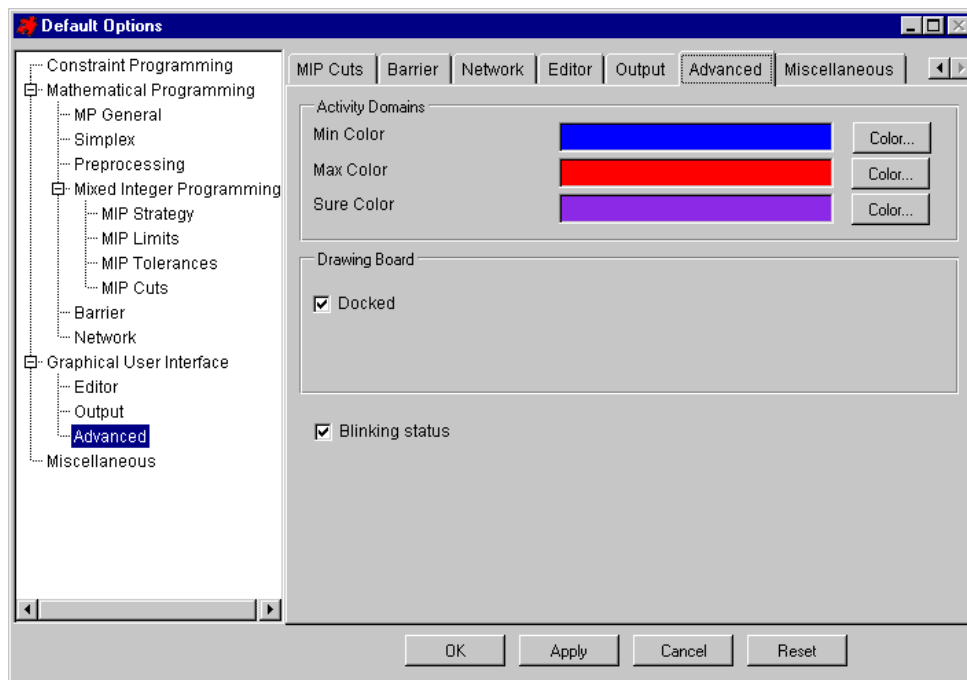
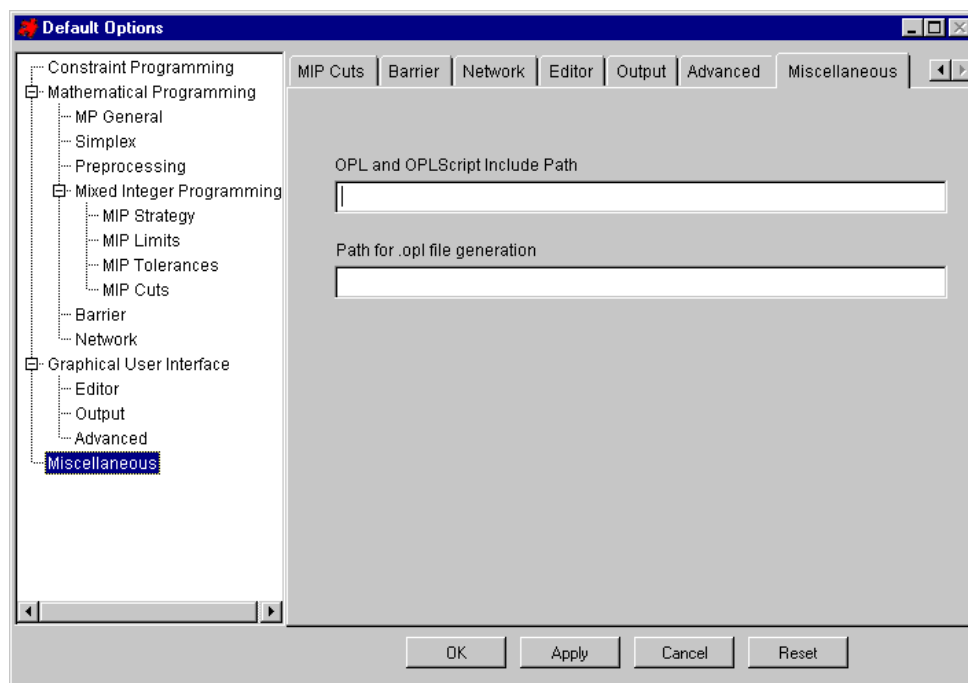


Figure 8.6 Advanced Options

- ◆ Activity Domains:
  - **Min Color** – used to draw the min rectangle (start min, end min).
  - **Max Color** – used to draw the max rectangle (start max, end max).
  - **Sure Color** – used to draw the surely overlapped time window (start min, end max)
- ◆ Drawing Board:
  - Docked** – the drawing board window is docked by default. Uncheck this box if you prefer the drawing board to float.
  - ◆ **Blinking status** – the yellow patch indicating a Waiting state blinks by default. Uncheck this box if you do not want the patch to blink in waiting mode. This modification is not taken into account in the Project Options, only in the Default Options.

## Setting Miscellaneous Options

Use the Miscellaneous notebook page to set the paths for OPL and OPLScript files, and for compiled models.



**Figure 8.7** *Miscellaneous Options*

### ◆ OPL and OPLScript Include Path

The pathname entered in this field indicates where to find files referenced in OPL or OPLScript code.

For a script that references model and data files, as in

```
Model car ("car.mod", "car.dat");
```

you can set this field to:

```
<OPLDIR>/examples/opl
```

to specify where to find the `car.mod` and `card.dat` files.

The only valid separator between pathnames is a semi-colon (;). Do not use blanks.

For a model that references a data file, as in `queens1.mod`:

```
int    n < "n.txt";
```

you can set the parameter to:

```
<OPLDIR>/examples/opl
```

to specify where to find the `n.txt` file.

◆ **Path for .opl file generation**

The pathname entered in this field indicates where `.opl` files are produced after generating a compiled model. See Chapter 12, *Generating Compiled Models*.



## Mathematical Programming

When customizing OPL Studio default or project options, you can select options for mathematical programming.

OPL parameters are identical to CPLEX parameters, except that OPL parameter names do not begin with a prefix, and contain mixed upper and lower case characters. For example, the two versions of the time limit parameter are as follows:

<code>tiLimit</code>	(OPL)
<code>CPX_PARAM_TILIMIT</code>	(CPLEX)

The default values are those displayed when the Default Options window appears.

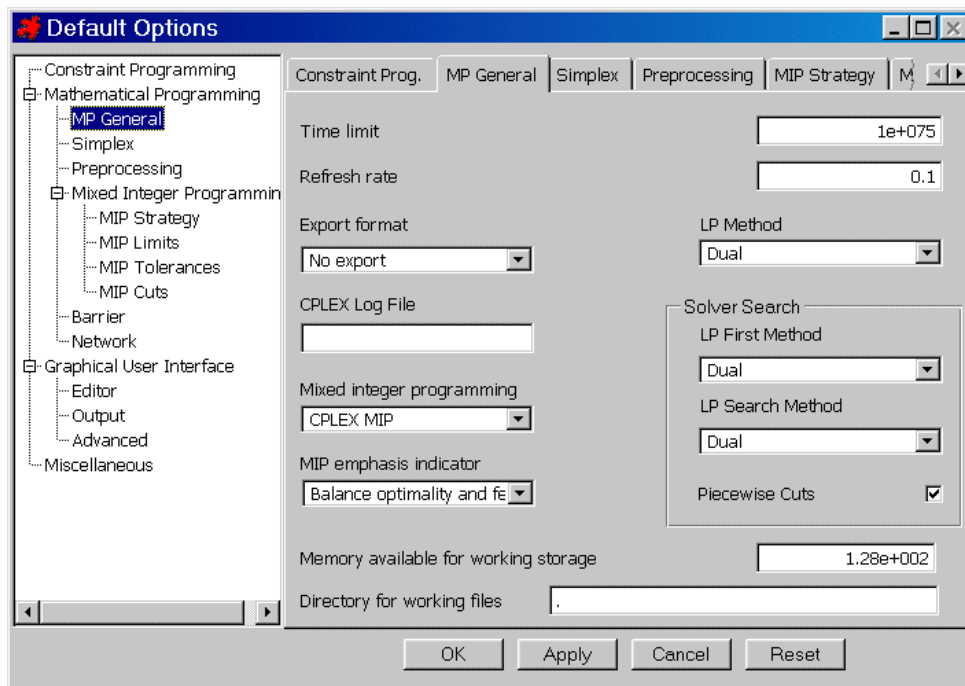
To set your options for the current session, click Apply or OK.

To reset the default values, click Reset.

Default options are saved in the `default.prj` file when you quit an application. This file is stored in your home directory on UNIX platforms and in your profile directory on Windows platforms. The options will be restored the next time you start OPL Studio. Only an option whose value differs from its default value is saved in the `default.prj` file.

Appendix A, *OPL Parameters*, contains an alphabetical list of the OPL parameters with their types and values.

## MP General



### ◆ Time limit

Sets the OPL parameter `tiLim`.

Sets the maximum time, in seconds, for computations before termination. The time limit applies to primal simplex, dual simplex, barrier, and mixed integer optimizations, as well as infeasibility finder computations. (Network simplex and barrier crossover operations are exceptions; these processes do not terminate if the time limit is exceeded.) The time limit includes preprocessing time. For ‘hybrid’ optimizations (such as network optimization followed by dual or primal simplex, barrier optimization followed by crossover), the cumulative time applies.

The value can be any positive number.

Default:  $1e^{75}$



### ◆ Refresh rate

Sets the screen refresh rate. Enter a decimal value, for example:

0.5 = refresh every half second

1 = refresh every second

0 = refresh as often as possible

Default: 0.1

### ◆ Export format

No export (default)

LP format

MPS format

SAV format

RLP format

REW format

These options apply to LP and MIP models. The exported file is created with its extension after running the model. The destination directory is the directory in which the project file is located, or the model file is located.

### ◆ CPLEX Log File

Sets the OPL parameter `cpexLogFile` used to generate the CPLEX log file.

The default value is empty, otherwise the value is a string containing a file name.

As an alternative to using the GUI, you can use the `setting` keyword in the model:

```
setting cplexLogFile = "myLogFile.log";
```

### ◆ Mixed integer programming

In this field you select the search method to be used for mixed integer programming.

CPLEX MIP (default)

Solver MIP	Uses an ILOG Solver search procedure (by default, a depth-first branch and bound search) with a linear relaxation.
------------	--

Solver	Uses ILOG Solver only (no linear relaxation). This option is valid only for integer programs and is ignored otherwise.
--------	--

### ◆ MIP emphasis indicator

Sets the OPL parameter `MIPEmphasis`.

In this field, specify whether CPLEX should use feasibility, optimality or a balance between searching for feasible solutions and proving optimality. The best bound choice emphasizes moving the best bound as an aggressive technique for proving optimality on extremely difficult models. Most models will reach the optimal solution fastest using one of the other choices, depending on the user's needs. The values are:

Balance optimality and feasibility (default)

Emphasize feasibility over optimality

Emphasize optimality over feasibility

Emphasize moving best bound

Emphasize hidden feasibles

### ◆ LP Method

In this field, you select a method to be used by OPL Studio to optimize a linear program. The available methods are:

Dual (default)

Primal

Network Primal

Network Dual

Barrier Primal

Barrier Dual

Barrier

### ◆ Solver Search

#### ● LP First Method

Here you select a value to be passed to the first LP solving method. Select from:

Primal (default)

Dual

Network Primal

Network Dual

Barrier Primal

Barrier Dual

- **LP Search Method**

Here you select a value to be passed to the LP incremental solving method. Select from:

Dual (default)

Primal

- **Piecewise Cuts**

When this box is checked (default), the OPL parameter `PiecewiseCuts` is set to on.

If the parameter is set to off, the default OPL search procedure will branch on the segments of the piecewise linear expression appearing in the problem statement.

If the parameter is set to on, in addition to the behavior described above, OPL can generate cuts to reduce the search space.

- ◆ **Memory available for working storage**

Sets the OPL parameter `workMem`.

Specifies an upper limit on the amount of central memory, in megabytes, that CPLEX is permitted to use for working files. See also `workDir`, below.

The value can be any positive number.

Default: 128

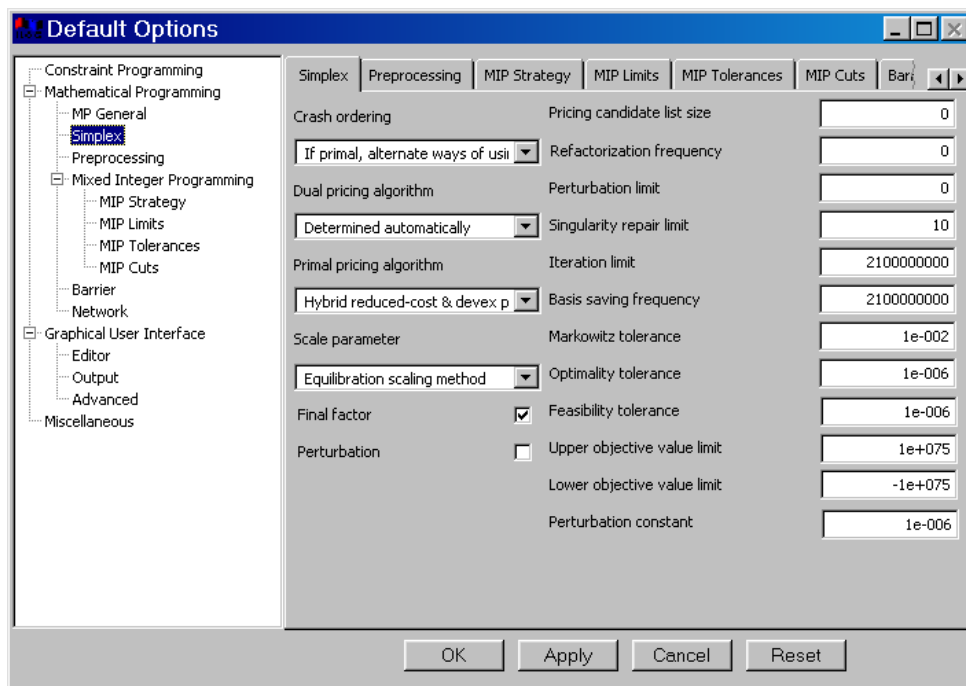
- ◆ **Directory for working files**

Sets the OPL parameter `workDir`.

Specifies the name of an existing directory in which CPLEX may store temporary working files, such as for MIP node files or for out-of-core barrier.

Default: . (a dot)

## Optimization Using Simplex



### ◆ Crash ordering

Sets the OPL parameter `craInd`.

Determines how CPLEX orders variables relative to the objective function when selecting an initial basis. Select a value from:

If primal, alternative ways of using objective coefficients; else, aggressive starting basis

If primal, ignore object coefficients during crash; else, aggressive starting basis

If primal, alternative ways of using objective coefficients; else, default starting basis

### ◆ Dual pricing algorithm

Sets the OPL parameter `dPriInd`.

The default pricing (Determined automatically) usually provides the fastest solution time, but many problems benefit from alternative settings. Select a value from:

- Determined automatically (default)
- Standard dual pricing
- Steepest-edge pricing
- Steepest-edge pricing in slack space
- Steepest-edge pricing, unit initial norms
- Devex pricing

### ◆ Primal pricing algorithm

Sets the OPL parameter `pPriInd`.

The default pricing (Hybrid reduced-cost & devex pricing) usually provides the fastest solution time, but many problems benefit from alternative settings. Select a value from:

- Reduced-cost pricing
- Hybrid reduced-cost & devex pricing (default)
- Devex pricing
- Steepest-edge pricing
- Steepest-edge pricing with slack initial norms
- Full pricing

### ◆ Scale parameter

Sets the OPL parameter `scaInd`.

Defines the method to be used for scaling the problem matrix. Select a value from:

- No scaling
- Equilibration scaling method (default)
- More aggressive scaling

### ◆ Final factor

Sets the OPL parameter `finalFactor`.

When preprocessing changes the model prior to optimization, a reverse operation (uncrush) occurs at termination to restore the full model with its solution. With default settings, the simplex optimizers perform a final basis factorization on the full model

before terminating. If you turn off this parameter, the final factorization after uncrushing will be skipped; on large models this can save some time, but computations that require a factored basis after optimization (for example, for the computation of the condition number  $\kappa$ ) may be unavailable, depending on the operations performed during preprocessing. If you run out of memory at the end of a simplex optimization, consider turning off final factorization. The values are:

0 Off

1 On (default)

#### ◆ Pricing candidate list size

Sets the OPL parameter `priceLim`.

Determines the maximum number of variables kept in the pricing candidate list. The value can be:

0 Determined automatically (default)

or any positive integer

#### ◆ Refactorization frequency

Sets the OPL parameter `reInv`.

Determines the number of iterations between refactorizations of the basis matrix. The value can be:

0 Determined automatically (default)

or any positive integer

#### ◆ Perturbation limit

Sets the OPL parameter `perLim`.

Determines the number of stalled iterations before perturbation will be performed. The value can be:

0 Determined automatically (default)

or any positive integer

#### ◆ Singularity repair limit

Sets the OPL parameter `singLim`.

Restricts the number of attempts to repair the basis when singularities are encountered. Once this limit is exceeded, CPLEX replaces the current basis with the best factorable basis that has been found.

The value can be any positive number.

Default: 10

### ◆ Iteration limit

Sets the OPL parameter `itLim`.

Determines the maximum number of iterations to be performed before the algorithm terminates without reaching optimality.

The value can be any positive integer.

Default: 2 100 000 000

### ◆ Basis saving frequency

Sets the OPL parameter `basInterval`.

Establishes the number of iterations between simplex basis file writings.

The value can be any positive integer.

Default: 2100000000

### ◆ Markowitz tolerance

Sets the OPL parameter `epMrk`.

Influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution.

The value can be any number between 0.0001 and 0.99999.

Default:  $1e^{-02}$

### ◆ Optimality tolerance

Sets the OPL parameter `epOpt`.

Influences the reduced-cost tolerance for optimality. This parameter governs how closely CPLEX must approach the theoretically optimal solution.

The value can be any number between  $1e^{-09}$  and  $1e^{-04}$

Default:  $1e^{-06}$

### ◆ Feasibility tolerance

Sets the OPL parameter `epRHS`.

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. 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  $1e^{-09}$  and  $1e^{-04}$

Default:  $1e^{-06}$

### ◆ Upper objective value limit

Sets the OPL parameter `objULim`.

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.

The value can be any number.

Default:  $1e^{75}$

### ◆ Lower objective value limit

Sets the OPL parameter `objLLim`.

Causes 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.

The value can be any number.

Default:  $-1e^{75}$

### ◆ Perturbation constant

Sets the OPL parameter `epPer`.

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.

The value can be any positive number  $\geq 1e^{-8}$

Default:  $1e^{-6}$

### ◆ Perturbation

Sets the OPL parameter `perInd`.

Indicates whether or not simplex perturbation is turned on.

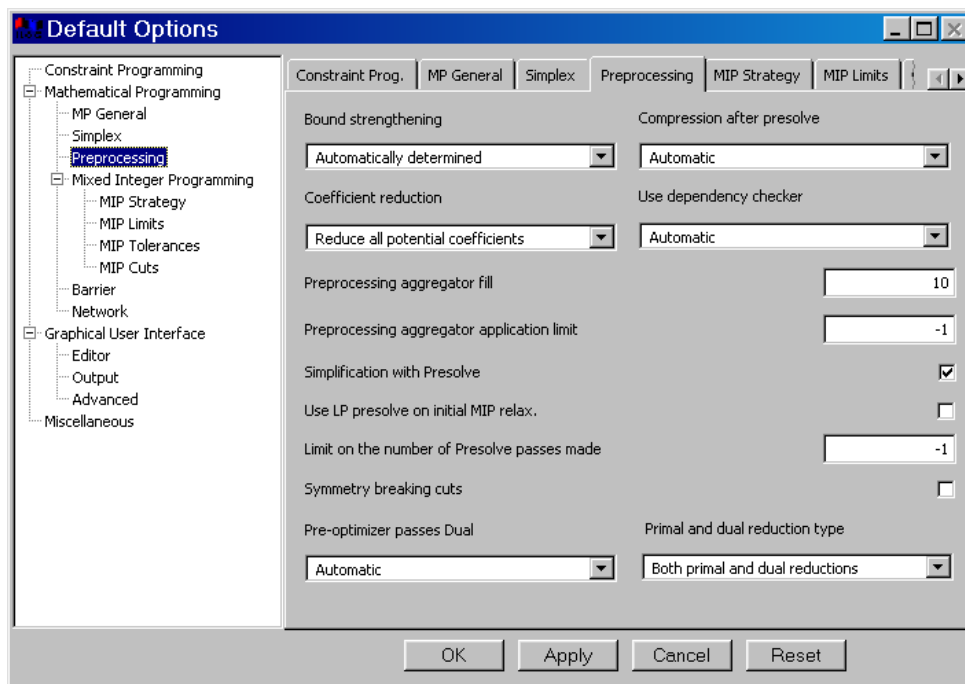
Simplex perturbation deals with situations – called stalling – in which no progress has been made in the objective function over a significant number of iterations.

If this box is not checked, CPLEX determines dynamically, during solution, whether progress is slow enough to merit a perturbation.

If this box is checked, all problems will be automatically perturbed as optimization begins. The situations in which this is useful will be rare and restricted to problems that exhibit extreme degeneracy.



## Preprocessing



### ◆ Bound strengthening

By default, the OPL parameter `bndStrenInd` is set to on and OPL Studio will use bound strengthening when solving MIP problems.

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 may take a long time due to its iterative nature. The value can be:

Automatically determined (default)

Do not apply bound strengthening

Apply bound strengthening

### ◆ **Compression of model after presolve**

Sets the OPL parameter `preCompress`.

Specifies whether CPLEX should compress the original model after presolve is performed. Compressing can save considerable storage space for large models. Under the automatic setting, CPLEX will decide whether to perform the compression based on model characteristics. The values are:

- Off
- Automatic (default)
- On

### ◆ **Coefficient reduction**

Sets the OPL parameter `coeRedInd`.

Determines how coefficient reduction will be used. Coefficient reduction improves the objective value of the initial (and subsequent) LP relaxations solved during branch-and-bound by reducing the number of non-integral vertices. Select a value from:

- Do not use
- Reduce only to integral coefficients
- Reduce all potential coefficients (default)

### ◆ **Symmetry**

Sets the OPL parameter `symmetry` to determine whether or not symmetry breaking cuts may be added, during the preprocessing phase, to a MIP model. The values are:

- 0 Off (default)
- 1 On

### ◆ **Preprocessing aggregator fill**

Sets the OPL parameter `aggFill`.

Limits variable substitutions by the aggregator. If the net result of a single substitution is more nonzeros than this value, the substitution will not be made.

The value can be any positive number.

Default: 10

### ◆ **Preprocessing aggregator application limit**

Sets the OPL parameter `aggInd`.

Invokes the aggregator to use substitution where possible to reduce the number of rows and columns before the problem is solved. If set to a positive value, the aggregator will be applied the specified number of times or until no more reductions are possible. The value can be:

- 1 Automatic (1 for LP, infinite for MIP)
  - 0 Do not use any aggregator
- Default: -1

### ◆ Simplification with Presolve

By default, the OPL parameter `preInd` is set to on.

Invokes CPLEX Presolve to simplify and reduce problems.

Uncheck the box to turn off this function.

### ◆ Use LP Presolve on initial MIP relaxation

Sets the OPL parameter `relaxPreInd`.

If you check this box, OPL Studio will invoke CPLEX Presolve for the initial relaxation of MIP problems.

### ◆ Limit on the number of Presolve passes made

Sets the OPL parameter `prePass`.

When set to a nonzero value, will invoke the CPLEX Presolve to simplify and reduce problems. When set to a positive value, the Presolve will be applied the specified number of times, or until no more reductions are possible. At the default value of -1, Presolve should continue only if it seems to be helping. The value can be:

-1 Determined automatically (default)

0 Do not use Presolve

or any positive integer

### ◆ Use dependency checker

Sets the OPL parameter `depInd`.

The dependency check strengthens problem reduction by detecting redundant constraints. Such reductions are usually most effective with the Barrier optimizer, but these reductions can be applied to LP or MIP problems. The settings for this parameter enable a user to control dependency checking.

-1 Automatic (default)

0 Off

1 On at beginning of preprocessing

2 On at end of preprocessing

3 On at beginning and end of preprocessing

If the dependency check is activated, it will search for dependent rows during preprocessing. If it is not activated, 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.

**◆ Pre-optimizer passes Dual**

Sets the OPL parameter `preDual`.

Determines whether CPLEX Presolve should pass the primal or dual linear programming problem to the linear programming optimization algorithm. By default, CPLEX chooses automatically. If the DUAL indicator is set to On, the CPLEX presolve algorithm is 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. The value can be:

Off

Automatic (default)

On

**◆ Primal and dual reduction type**

Sets the OPL parameter `reduce`.

Determines whether primal reductions, dual reductions, or both, are performed during preprocessing. The value can be:

No primal and dual reductions

Only primal reductions

Only dual reductions

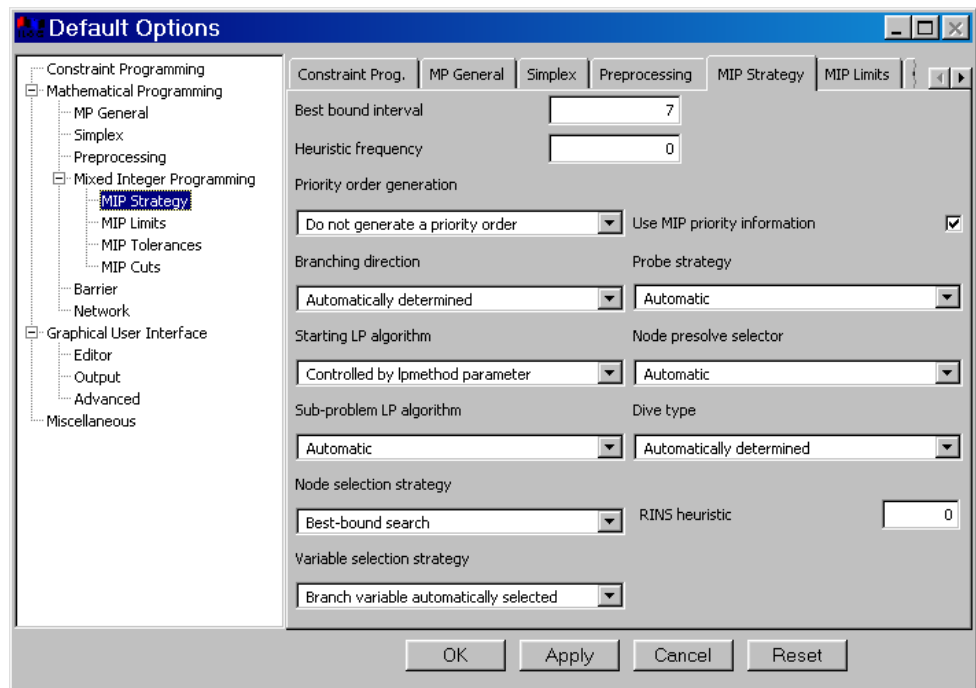
Both primal and dual reductions (default)

## Mixed Integer Programming

There are four notebook pages for mixed integer programming (MIP):

- ◆ MIP Strategy
- ◆ MIP Limits
- ◆ MIP Tolerances
- ◆ MIP Cuts

### MIP Strategy



◆ **Best bound interval**

Sets the OPL parameter `BBInterval`.

When `nodeSel=2`, the value of `BBInterval` is the interval at which the best bound node, instead of the best estimate node, will be selected from the tree. The value can be:

0 Best estimate node always selected

1 Best bound node always selected

or, any positive integer

Default: 7

◆ **Heuristic frequency**

Sets the OPL parameter `heurFreq`.

Determines how often to apply the periodic heuristic. The value can be:

0 Do not apply heuristic at nodes (default)

or any positive integer

◆ **Priority order generation**

Sets the OPL parameter `MIPOrdType`. Select a value from:

Do not generate a priority order (default)

Use decreasing cost

Use increasing bound range

Use increasing cost per coefficient count

◆ **Branching direction**

Sets the OPL parameter `brDir`. Select a value from:

Down branch first

Automatically determined (default)

Up branch first

### ◆ Starting LP algorithm

Sets the OPL parameter `startAlg` to determine which LP algorithm should be used to solve the initial relaxation of the MIP. Select a value from:

Controlled by the `lpmethod` parameter (default)

Primal simplex

Dual simplex

Network optimizer followed by dual simplex

Barrier with crossover

### ◆ Sub-problem LP algorithm

Sets the OPL parameter `subAlg`, the algorithm to be used on MIP subproblems. Select a value from:

Automatic (default)

Primal simplex

Dual simplex

Network optimizer followed by dual simplex

Barrier with crossover

### ◆ Node selection strategy

Sets the OPL parameter `nodeSel`. Select a value from:

Depth-first search

Best-bound search (default)

Best-estimate search

Alternative best-estimate search

### ◆ Variable selection strategy

Sets the OPL parameter `varSel`. Select a value from:

Branch on variables with minimum infeasibility

Branch variable automatically selected (default)

Branch on variable with maximum infeasibility

Branch based on pseudo cost

Strong branching

Branch based on pseudo reduced cost

### ◆ Use MIP priority information

By default, the OPL parameter `MIPOrdInd` is set to on and OPL Studio will use priority order information (if it exists) for the next MIP optimization.

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).

Uncheck the box to switch off this function.

### ◆ Probe strategy

Sets the OPL parameter `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 1 can result in dramatic reductions or dramatic increases in solution time on particular models. The value can be:

No probing

Automatic (default)

Probing level 1

Probing level 2

Probing level 3

### ◆ Node presolve selector

Sets the OPL parameter `preslNd`.

Indicates whether node presolve should be performed at the nodes of a mixed integer programming solution. Node presolve can significantly reduce solution time for some models. The default setting is generally effective at determining whether to apply node presolve, although runtimes can be reduced for some models by turning node presolve off. The value can be:

No node presolve

Automatic (default)

Force node presolve



### ◆ Dive type

Sets the OPL parameter `diveType` to determine the MIP dive strategy.

The MIP traversal strategy occasionally performs probing dives, where it looks ahead at both children nodes before deciding which node to choose. The values are:

Automatically determined (default)

Traditional dive

Probing dive

Guided dive

The default setting lets CPLEX choose when to perform a probing dive. The option Traditional dive directs CPLEX never to perform probing dives, Probing dive to always perform probing dives, and Guided dive to spend more time exploring potential solutions that are similar to the current incumbent.

### ◆ RINS heuristic

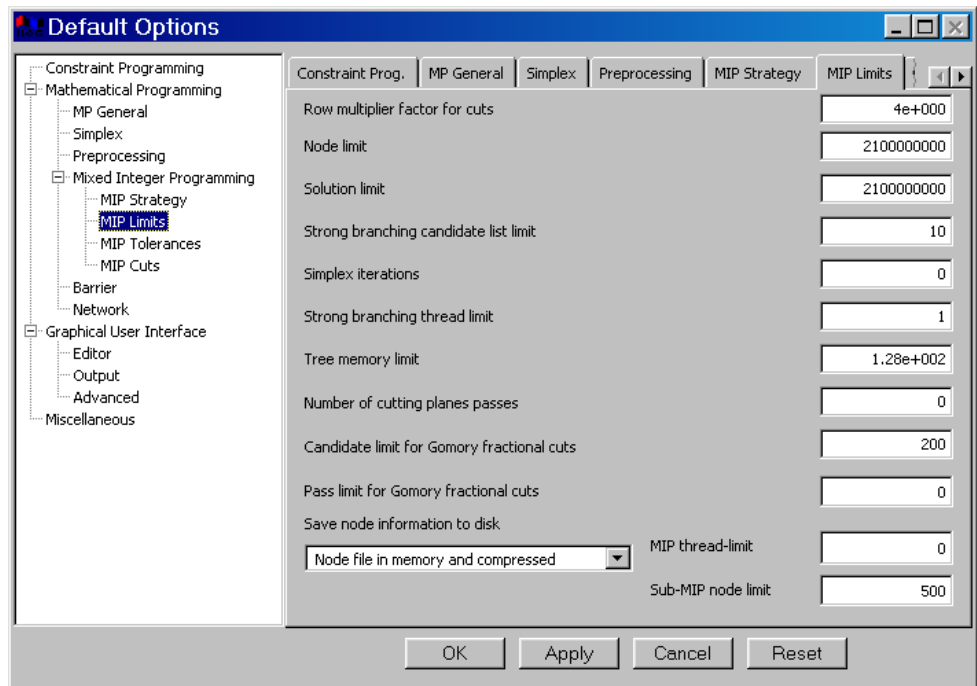
Sets the OPL parameter `RINSHeur` to determine how often to apply the relaxation induced neighborhood search (RINS) heuristic. The values are:

-1 Turns off the RINS heuristic

0 Applies the RINS heuristic at an interval chosen automatically by CPLEX (default)

Any positive integer to apply the RINS heuristic at the requested node interval. For example, setting `RINSHeur` to 20 indicates that the RINS heuristic will be called at nodes 0, 20, 40, 60, etc.

## MIP Limits



### ◆ Row multiplier factor for cuts

Sets the OPL parameter `cutsFactor`.

Limits the number of both clique and cover cuts that can be added. The number of rows in the problem with cuts added is limited to this value times the original number of rows.

The value can be any positive number.

Default: 4

### ◆ Node limit

Sets the OPL parameter `nodeLim`.

Determines the maximum number of nodes solved before the algorithm terminates, without reaching optimality.

The value can be any positive integer.

Default: 2 100 000 000

◆ **Solution limit**

Sets the OPL parameter `intSolLim`.

Determines the number of MIP solutions to be found before stopping.

The value can be any positive integer.

Default: 2 100 000 000

◆ **Strong branching candidate list limit**

Sets the OPL parameter `strongCandLim`.

Controls the length of the candidate list when using the "strong branching" variable selection setting.

The value can be any positive number.

Default: 10

◆ **Simplex iterations**

Sets the OPL parameter `strongItLim`.

Controls the number of simplex iterations performed on each variable in the candidate list when using the "strong branching" variable selection setting.

The value can be any positive number.

The default setting 0 chooses the iteration limit automatically.

◆ **Strong branching thread limit**

Sets the OPL parameter `strongThreadLim`.

Controls the number of parallel threads used to perform strong branching. Note that this parameter does nothing if the MIP thread limit is greater than 1.

The value can be any positive number.

Default: 1

◆ **Tree memory limit**

Sets the OPL parameter `treLim`.

Sets an upper limit on the amount of memory (in megabytes) that the branch-and-bound tree can consume. The action taken by CPLEX, when the amount of memory required to store the branch-and-bound information exceeds the `treLim` parameter, depends on the setting of the node file parameter (`nodeFileInd`). If the parameter is set to 0 (no node file), CPLEX terminates optimization. Otherwise, CPLEX continues optimization, transferring nodes from the branch-and-bound tree to node files to keep the required memory below the tree memory limit.

The value can be any non-negative number.

Default: 128

◆ **Number of cutting plane passes**

Sets the OPL parameter `cutPass`.

Sets the upper limit on the number of passes CPLEX performs when generating cutting planes on a MIP model. The value can be:

-1 None

0 Automatically determined (default)

Any positive integer, to indicate the number of passes to perform

◆ **Candidate limit for Gomory fractional cuts**

Sets the OPL parameter `fracCand`.

Limits the number of candidate variables for generating Gomory fractional cuts.

The value can be any non-negative integer.

Default: 200

◆ **Pass limit for Gomory fractional cuts**

Sets the OPL parameter `fracPass`.

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, `fracCuts`, is set to a nonzero value.

The value can be any positive integer.

Default: 0

### ◆ Save node information to disk

Sets the OPL parameter `nodeFileInd`.

Used when the tree memory limit (set by `treLim`) is reached. If the node file parameter is set to zero when the tree memory limit is reached, optimization is terminated. Otherwise, a group of nodes is removed from the in-memory set as needed. By default, CPLEX transfers nodes to node files when the in-memory set is larger than 128 MBytes, and it keeps the resulting node ‘files’ in compressed form in memory.

The use of node files is described in more detail in the *CPLEX User’s Manual*.

The value can be:

No node file

Node file in memory and compressed (default)

Node file on disk

Node file on disk and compressed

### ◆ MIP thread-limit

Sets the OPL parameter `MIPThreads`.

0 Limit determined by global thread (default)

>0 Upper limit on threads for Parallel MIP

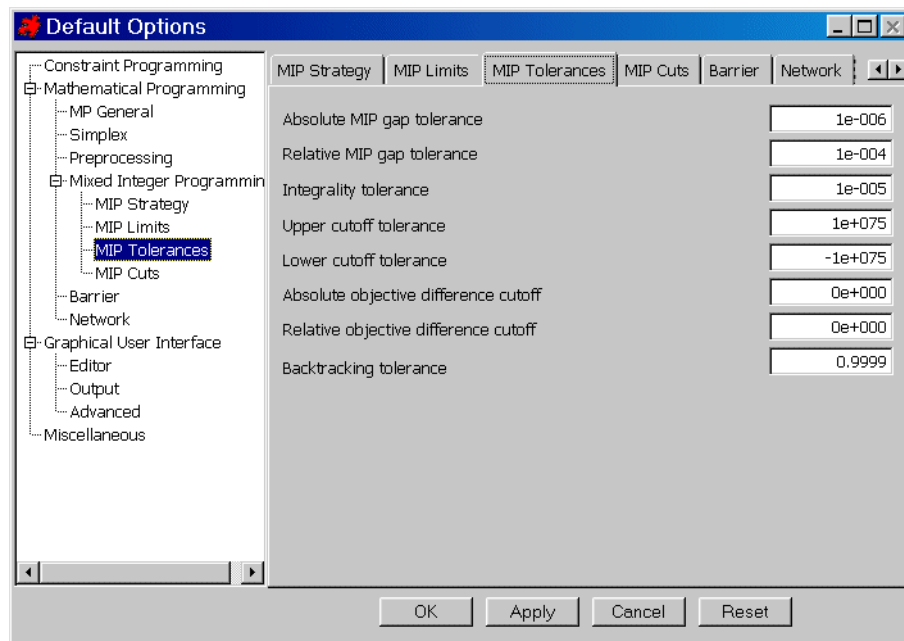
### ◆ Sub-MIP node limit

Sets the OPL parameter `subMIPNodeLim` to define the MIP subnode limit. This parameter restricts the number of nodes searched during application of the relaxation induced neighborhood search (RINS) heuristic. See the `RINSHeur` parameter.

The value can be any positive integer.

Default: 500

## MIP Tolerances



### ◆ Absolute MIP gap tolerance

Sets the OPL parameter `epAGap`.

Defines an absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When the difference falls below the value of this parameter, the MIP optimization is stopped.

The value can be any positive number.

Default:  $1e^{-06}$

### ◆ Relative MIP gap tolerance

Sets the OPL parameter `epGap`.

Defines a relative tolerance on the gap between the best integer objective and the object of the best node remaining. When the value  $|bestnode - bestinteger| / (1e-10 + |bestinteger|)$  falls below this value, the MIP optimization is stopped.

The value can be any number between 0 and 1.

Default:  $1e^{-04}$

### ◆ Integrality tolerance

Sets the OPL parameter `epInt`.

Specifies the amount by which an integer variable can be different from an integer and still be considered feasible.

The value can be any number between  $1e^{-09}$  and 1.

Default:  $1e^{-05}$

### ◆ Upper cutoff tolerance

Sets the OPL parameter `cutUp`.

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

The value can be any number.

Default:  $1e^{75}$

### ◆ Lower cutoff tolerance

Sets the OPL parameter `cutLo`.

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

The value can be any number.

Default:  $-1e^{75}$

### ◆ Absolute objective difference cutoff

Sets the OPL parameter `objDif`.

Used to update the cutoff each time a MIP solution is found. This absolute value will be subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the MIP optimization to ignore integer solutions that are not at least this amount better than the one found so far. This parameter 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. Negative values for this parameter 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.

The value can be any number.

Default: 0

### ◆ Relative objective difference cutoff

Sets the OPL parameter `relObjDif`.

Used to update the cutoff each time a MIP 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 MIP optimization to ignore integer solutions that are not at least this amount better than the one found so far. This parameter 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 `relObjDif` and `objDif` are nonzero, the value of `objDif` will be used.

The value can be any positive number.

Default: 0

### ◆ Backtracking tolerance

Sets the OPL parameter `btTol`.

Controls how often backtracking is done during the branching process. The decision when to backtrack depends on three values that change during the course of the optimization:

- the objective function value of the best integer feasible solution (“incumbent”)
- the best remaining objective function value of any unexplored node (“best node”)
- the objective function value of the most recently solved node (“current objective”).

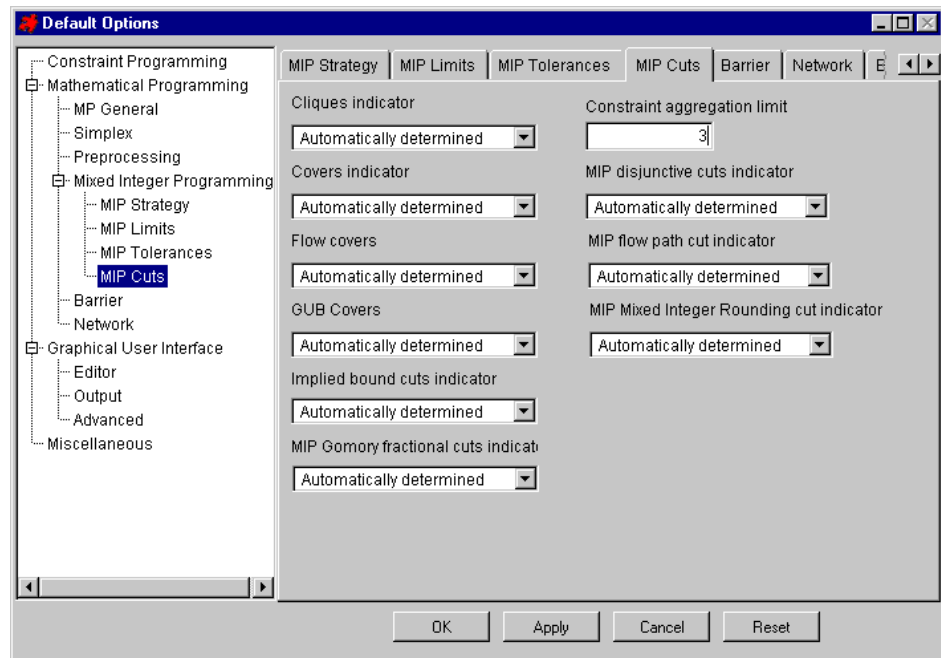
If a cutoff tolerance (see `cutUp` and `cutLo`) has been set by the user, then that value is used as the incumbent until an integer feasible solution is found. The “target gap” is defined to be the absolute value of the difference between the incumbent and the best node, multiplied by this backtracking parameter. CPLEX does not backtrack until the absolute value of the difference between the current objective and the best node is at least as large as the target gap. Low values of this backtracking parameter thus tend to increase the amount of backtracking, which makes the search process more of a pure best-bound search. Higher parameter values tend to decrease backtracking, making the search more of a pure depth-first search. The backtracking value has effect only after an integer feasible solution is found or when a cutoff has been specified. Note that this backtracking value merely permits backtracking but does not force it; CPLEX may choose to continue searching a branch of the tree if it seems a promising candidate for finding an integer feasible solution.

The value can be any number between 0 and 1.

Default: 0.9999



## MIP Cuts



### ◆ Cliques indicator

Sets the OPL parameter `cliques`. Select a value from:

- Do not generate clique cuts
- Automatically determined (default)
- Generate clique cuts moderately
- Generate clique cuts aggressively

### ◆ Covers indicator

Sets the OPL parameter `covers`. Select a value from:

- Do not generate cover cuts
- Automatically determined (default)
- Generate cover cuts moderately
- Generate cover cuts aggressively

◆ **Flow covers**

Sets the OPL parameter `flowCovers`.

Determines whether or not to generate flow cuts for the problem. Setting the value to 0, the default, indicates that the attempt to generate flow cuts should continue only if it seems to be helping. The value can be one of the following:

- Do not generate flow cuts
- Automatically determined (default)
- Generate flow cuts moderately
- Generate flow cuts aggressively

◆ **GUB covers**

Sets the OPL parameter `gubCovers`.

Determines whether or not to generate GUB cuts for the problem. Setting the value to 0, the default, indicates that the attempt to generate GUB cuts should continue only if it seems to be helping. The value can be:

- Do not generate GUB cuts
- Automatically determined (default)
- Generate GUB cuts moderately
- Generate GUB cuts aggressively

◆ **Implied bound cuts indicator**

Sets the OPL parameter `implBd`.

Determines whether or not to generate implied bound cuts for the problem. Setting the value to 0, the default, indicates that the attempt to generate implied bound cuts should continue only if it seems to be helping. The value can be:

- Do not generate implied bound cuts
- Automatically determined (default)
- Generate implied bound cuts moderately
- Generate implied bound cuts aggressively

◆ **MIP Gomory fractional cuts indicator**

Sets the OPL parameter `fracCuts` to determine whether or not Gomory fractional cuts should be generated. The value can be:

- 1 Do not generate Gomory fractional cuts
- 0 Automatically determined (default)
- 1 Generate Gomory fractional cuts moderately
- 2 Generate Gomory fractional cuts aggressively

◆ **Constraint aggregation limit**

Sets the OPL parameter `aggCutLim` to limit the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts.

The value can be any non-negative integer.

Default: 3

◆ **MIP disjunctive cuts indicator**

Sets the OPL parameter `disjCuts` to determine whether or not disjunctive cuts should be generated for the problem. Setting the value to 0, the default, indicates that the attempt to generate disjunctive cuts should continue only if it seems to be helping. The value can be:

- 1 Do not generate disjunctive cuts
- 0 Automatically determined (default)
- 1 Generate disjunctive cuts moderately
- 2 Generate disjunctive cuts aggressively

◆ **MIP flow path cuts indicator**

Sets the OPL parameter `flowPaths` to determine whether or not flow path cuts should be generated for the problem. Setting the value to 0, the default, indicates that the attempt to generate flow path cuts should continue only if it seems to be helping. The value can be:

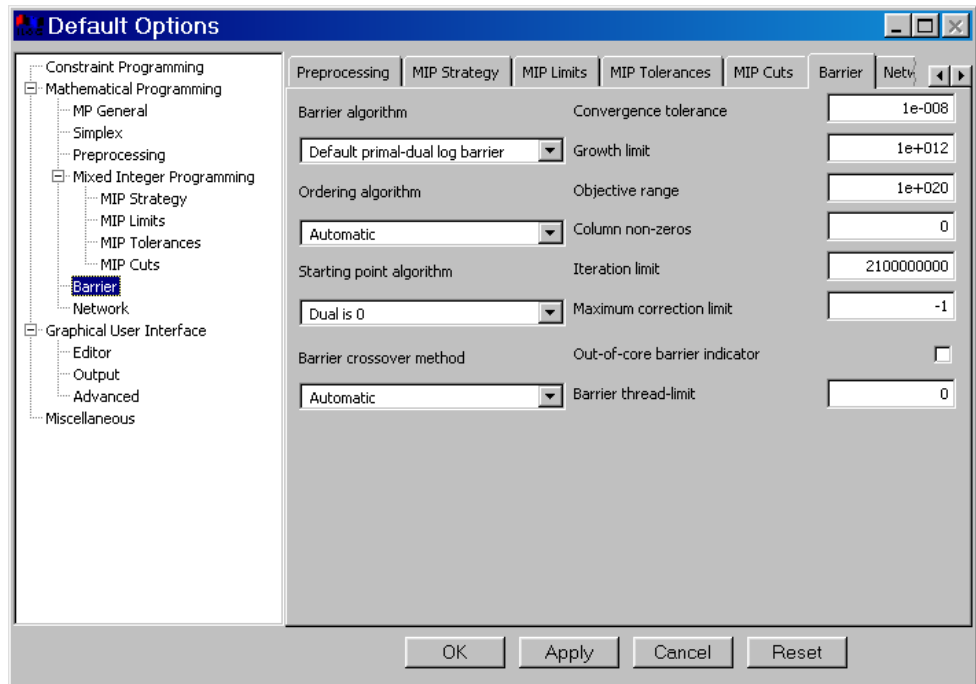
- 1 Do not generate flow path cuts
- 0 Automatically determined (default)
- 1 Generate flow path cuts moderately
- 2 Generate flow path cuts aggressively

**◆ MIP Mixed Integer Rounding cuts indicator**

Sets the OPL parameter `MIRCuts` to determine whether or not to generate MIR cuts for the problem. Setting the value to 0, the default, indicates that the attempt to generate MIR cuts should continue only if it seems to be helping. The value can be:

- 1 Do not generate MIR cuts
- 0 Automatically determined (default)
- 1 Generate MIR cuts moderately
- 2 Generate MIR cuts aggressively

## Barrier Algorithm



### ◆ Barrier algorithm

Sets the OPL parameter `barAlg` to determine which barrier algorithm is used. The default value (Default primal-dual log barrier) is normally fastest, but the alternative settings may eliminate numerical difficulties relating to infeasibility. The value can be:

- Default primal-dual log barrier (default)
- Infeasibility-estimate start
- Infeasibility-constant start
- Standard barrier

### ◆ Ordering algorithm

Sets the OPL parameter `barOrder` to define the algorithm to be used to permute the rows of the constraint matrix in order to reduce fill in the Cholesky factor. The value can be:

- Automatic (default)
- Approximate minimum degree (AMD)
- Approximate minimum fill (AMF)
- Nested dissection (ND)

### ◆ Starting point algorithm

Sets the OPL parameter `barStartAlg` to define the algorithm to be used to compute the initial starting point for the barrier solver. The value can be:

- Dual is 0 (default)
- Estimate dual
- Average of primal estimate, dual 0
- Average of primal estimate, estimate dual

### ◆ Barrier crossover method

Sets the OPL parameter `barCrossAlg` to determine which, if any, crossover method is performed at the end of a Barrier optimization. The value can be:

- 1 No crossover
- 0 Automatic (default)
- 1 Primal crossover
- 2 Dual crossover

### ◆ Convergence tolerance

Sets the OPL parameter `barEpComp` to determine 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 value can be any positive number  $\geq 1e^{-10}$

Default:  $1e^{-8}$

### ◆ Growth limit

Sets the OPL parameter `barGrowth`. This parameter 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 value can be any positive number.

Default:  $1e^{12}$

### ◆ Objective range

Sets the OPL parameter `barObjRng` to specify the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems.

The value can be any positive number.

Default:  $1e^{20}$

### ◆ Column nonzeros

Sets the OPL parameter `barColNz`. This parameter is used in the recognition of dense columns. If columns in the presolved and aggregated problem exist with more entries than the given value, such columns will be considered dense and will be treated specially by CPLEX barrier to reduce their effect.

**Note:** *If the problem contains fewer than 400 rows, dense column handling will not be initiated.*

The value can be:

dynamically calculated

or any positive integer

Default: 0

◆ **Iteration limit**

Sets the OPL parameter `barItLim`.

Specifies the number of barrier iterations before termination. When set to 0, no barrier iterations occur, but problem “set up” occurs and information about the set up is displayed (such as Cholesky factorization information). The value can be:

0 No barrier iterations

or any positive integer

Default: 2 100 000 000

◆ **Maximum correction limit**

Sets the OPL parameter `barMaxCor` to specify the maximum number of centering corrections done on each iteration. An explicit value greater than 0 may improve the numerical performance of the algorithm at the expense of computation time. The value can be:

-1 Automatically determined (default)

0 None

or any positive integer

◆ **Out-of-core barrier indicator**

Sets the OPL parameter `barOOC` to specify whether the barrier optimizer should use out-of-core storage (on disk) for the Cholesky factorization. Disk usage is controlled by the parameters `workmem` and `workDir`.

By default this parameter is set to off. Check the box to use out-of-core storage.

◆ **Barrier thread limit**

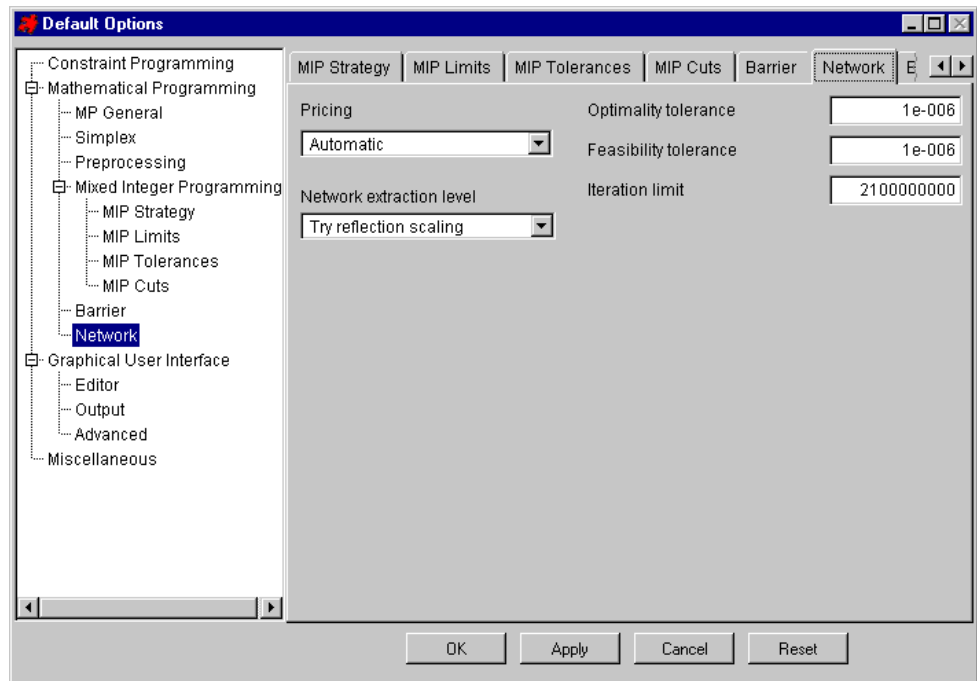
Sets the OPL parameter `barThreads`

0 Limit determined by global thread default

>0 Upper limit on threads for Parallel Barrier



## Network Simplex Algorithm



### ◆ Pricing

Sets the OPL parameter `netPPriInd`.

The default (Automatic) shows best performance for most problems, and currently is equivalent to 3. Select a value from:

- Automatic (default)
- Partial pricing
- Multiple partial pricing
- Multiple partial pricing with sorting

◆ **Network extraction level**

Sets the OPL parameter `netFind` to establish the level of network extraction for network simplex optimizations. Select a value from:

- Extract pure network only
- Try reflection scaling (default)
- Try general scaling

◆ **Optimality tolerance**

Sets the OPL parameter `netEpOpt`.

Settings 1 and 2 differ only during Phase I. Setting 2 shows monotonic values, whereas 1 usually does not.

The value can be any number from  $1e^{-4}$  to  $1e^{-11}$

Default:  $1e^{-6}$

◆ **Feasibility tolerance**

Sets the OPL parameter `netEpRHS`.

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 from  $1e^{-4}$  to  $1e^{-11}$

Default:  $1e^{-6}$

◆ **Iteration limit**

Sets the OPL parameter `netItLim` to determine the maximum number of iterations to be performed before the algorithm terminates without reaching optimality.

The value can be any non-negative integer.

Default: 2100000000

## Results of Mathematical Programming

Once you have selected your options and executed the Run command, the results are displayed in the CPLEX notebook page at the bottom of the Main window.

The screenshot shows the CPLEX notebook page with the following data:

Primal Phase I		Dual Phase I		Barrier		MIP	
Infeasibility		Infeasibility		Primal objective		Nodes	0
Iterations		Iterations				Nodes left	0
Primal Phase II		Dual Phase II		Dual objective		Iterations	
Objective		Objective	1.8e+003			Best	0
Iterations		Iterations	2			Bound	
Primal Crossover		Dual Crossover		Constraints		Cutoff	
Push		Push			3		0
Exchange		Exchange		Variables	2		

Below the table, the phase indicator is set to "Dual phase II". At the bottom, the status bar shows "OPL Studio is idle: 1 solution(s) found" and the file path "...lexamplestopl\lgas.prj".

This page informs you of the phase that CPLEX is currently in.

- ◆ The long bar at the bottom of the notebook page is the phase indicator. The phase displayed may be one of the following:

Presolve

Network

Primal Phase I

Primal Phase II

Dual Phase I

Dual Phase II

Barrier

The various frames in the rest of the page give information pertaining to each specific phase.

- ◆ For primal and dual phase I:
  - Infeasibility measure
  - Number of iterations
- ◆ For primal and dual phase II:
  - Value of the objective function
  - Number of iterations
- ◆ For the primal and dual crossover:
  - Number of pushes
  - Number of exchanges
- ◆ For barrier:
  - Primal objective value
  - Dual objective value
  - Number of iterations
- ◆ For the MIP:
  - Number of nodes explored
  - Number of nodes still left to explore
  - Number of iterations
  - Best value found so far (upper bound)
  - Lower bound currently used for this subtree
  - Cutoff value
- ◆ This page also displays the:
  - Number of constraints
  - Number of variables

that are submitted to the linear solver.

## Working with a Database

This chapter explains how to use the database connection feature offered by ILOG OPL Studio. The example used is based on the bridge problem, which is a scheduling application discussed in detail in the *ILOG OPL Studio: Language Manual*, and presented in the distributed project `bridge.prj` located in the following subdirectory:

- ◆ For UNIX systems

```
<installation>/OPLSt37/examples/opl/scheduler
```

- ◆ For Windows XP, Windows 2000, Windows NT 4, and Windows 98

```
c:\ILOG\OPLSt37\examples\opl\scheduler
```

However, this time, for the purposes of the example shown here, the data items are stored in tables in a relational database. In this chapter, you will see how to:

- ◆ establish a connection to a database from OPL Studio
- ◆ read database relations into OPL sets
- ◆ create a new relational table from OPL
- ◆ write an OPL set to a database by inserting new rows into a table.

**Note:** OPL cannot read a double-byte string from a database. This means that all string data from a database must contain standard single-byte characters.

## Supported Databases

OPL Studio 3.7 interfaces with the RDBMS supported by ILOG DB Link 5.0. The specific database systems supported by each of the UNIX and Windows platforms are listed in the README file.

For example:

- ◆ On UNIX systems
  - Oracle 8.1 (Solaris, HP, Linux, RS6000)
- ◆ On Windows XP, Windows 2000, Windows NT 4, and Windows 98
  - Oracle 7.3, 8.0, 8.1, 9i

## Database Connectivity

### The Connection Command

The OPL instruction **DBconnection** establishes a connection to a database. It requires two arguments: the database client you want to use and the connection string.

The first argument is a string indicating the name of a database system as known by ILOG DB Link and must have a value such as `oracle81`. The complete list of possible values can be found in the README file.

**Note:** *Prior to using a database connection, you must ensure that the corresponding database client is correctly installed on your system.*

The second argument, the connection string, must comply with a format that depends on the target RDBMS. For Oracle, for example, the format is:

```
[<user>]/[<password>][@<SQL Net id>]
```

where the SQL Net id is:

@<net>:<hostname>[:<SID>] for SQL Net V1, and @<service name> for SQL Net V2.

In the example

```
DBconnection db("oracle81", "scott/tiger@ilog");
```

the user `scott` with the password `tiger` will connect to the Oracle database called `ilog`.

## The Environment Variable

### ◆ On UNIX systems

When integrating with your application you should make the `LD_LIBRARY_PATH` (or `SHLIB_PATH` or `LIBPATH`) environment variable point to the database driver location and to the ILOG DB Link driver directory.

The DB Link drivers are provided in directories of the type:

`<OPLDIR>/lib/<platform>/<shared_format>`

Details are provided in the README file.

ILOG OPL 3.7 is in theory independent of ILOG DB Link, since it comes with all the DB Link dynamic link libraries. In certain cases, however, DB Link drivers need to be relinked for the platform and database release that OPL needs to connect to. Please consult the ILOG DB Link documentation that explains how to rebuild the DB Link drivers.

In these specific cases:

- Install ILOG DB Link 5.0 and patches if necessary.
- Relink the DB Link drivers for the platform and RDBMS in question.
- Type the name of the ILOG DB Link dynamic library directory at the beginning of your `LD_LIBRARY_PATH` (or `SHLIB_PATH` or `LIBPATH`).

### ◆ On Windows XP, Windows 2000, Windows NT 4, and Windows 98

- To use database connectivity on Windows from the GUI, you do not need to set paths. The `dblink.ini` file is in the `bin` directory containing the `oplst.exe` file and calls the libraries in the directory `lib/msvc6/dll_mda`.

- To use database connectivity on Windows from an application that integrates the OPL library, you need to check that the dynamic link libraries required by DB Link can be found in the `PATH` variable. (Example for `msvc6/stat_mda`, COM or JNI applications:  
`set PATH=c:\ilog\OPLst37\lib\msvc6\dll_mda;%PATH%`)

Dynamic link libraries are provided with OPL Studio 3.7. These `dll` files are for `msvc6/stat_mda`, COM/ActiveX and Java. They can be found in `<OPLDIR>\lib\msvc6\dll_mda`. For the other ports, you need to install ILOG DB Link 5.0.

- In the case of deployment on another machine, make sure that the `PATH` variable of the target machine indicates where the dynamic drivers are located.

Consult the README file for details.

## Prerequisites

- ◆ In order to follow this example, a minimal knowledge of the syntax of the query language, SQL, would help.
- ◆ If you are working with ODBC, you are not required to have Microsoft Access installed on your computer. However, having this product installed will allow you to view the contents of the database file `abridge.mdb`.
- ◆ If you are working with Oracle:
  - you must have a user account and a password allowing you to connect to a pre-existing database
  - the Oracle client must be installed.

## The Bridge Example

For this example, you will need to use the following files from your release distribution:

- ◆ Windows XP, Windows 2000, Windows NT 4, and Windows 98

```
c:\ILOG\OPLSt37\examples\opl\database\abridge.mod
```

which is the Bridge example using a connection to ODBC

```
c:\ILOG\OPLSt37\examples\opl\database\abridge.mdb
```

which is a Microsoft Access database source containing the data for the Bridge example.

- ◆ UNIX systems

```
<installation-directory>/OPLSt37/examples/opl/database/obridge.mod
```

which is the Bridge example using a connection to Oracle 8.

```
<installation-directory>/OPLSt37/examples/opl/database/obridge.sql
```

which is an SQL script to help you load the data for the Bridge example into an Oracle database of your choice.

The Bridge example involves finding a schedule that minimizes the time needed to build a five-segment bridge.

The project contains a set of 46 tasks and a set of constraints within these tasks. Most tasks require a resource (e.g. a crane), and tasks requiring the same resource cannot overlap in time.

In addition, several other constraints must be satisfied, as explained in detail in the *ILOG OPL Studio: Language Manual*, Chapter 15.



The data items are organized in tables in a relational database. OPL establishes a connection to the database and initializes the model by reading the corresponding relational tables.

After the optimal schedule is computed, a new table is created and the optimal schedule is stored in this new table.

## Setting Up the Database

Before running the example, you must ensure that a database containing the data for the Bridge problem is properly installed and available on your particular system. To do this, follow these steps:

◆ On Windows XP, Windows 2000, Windows NT 4, and Windows 98

- Select: Start>Settings>Control Panel>ODBC
- On the page User DSN press the button Add...
- Select Microsoft Access driver (\*.mdb)
- In the field Data Source Name, enter abridge
- In the field Description, enter the comment OPL example
- Click on the button Select... and look for the file abridge.mdb in the directory  
c:\ILOG\OPLSt37\examples\database

◆ On UNIX

- Ensure that the environment variable ORACLE\_HOME is correctly set to the Oracle installation on your machine. For example:

```
$ setenv ORACLE_HOME /nfs/oracle/8/solaris
```

- Use the SQL statements from the file:

```
<installation-directory>/OPLSt37/examples/opl/database/obridge.sql
```

in order to load the data for the example into your database.

For example, you can launch the SQL shell \$ORACLE\_HOME/bin/sqlplus, then type:

```
SQL> @obridge.sql;
```

- Ensure that the environment variable LD\_LIBRARY\_PATH contains both the path \$ORACLE\_HOME/lib and the path <installation-directory>/OPLSt37/bin

## The Data Tables

The data for the Bridge example is stored in nine relational tables:

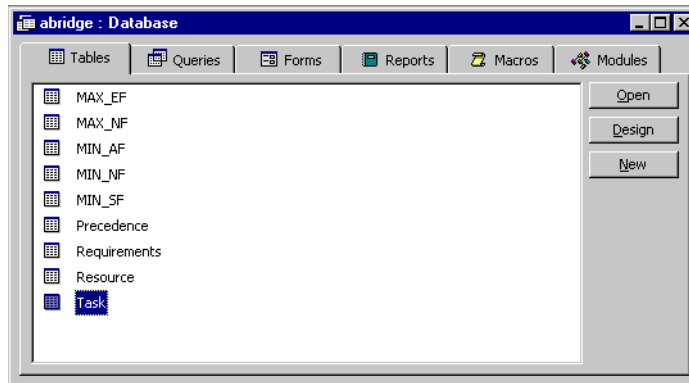
MAX\_EF, MAX\_NF, MIN\_AF, MIN\_NF, MIN\_SF, Precedence, Requirements, Resource, Task

- ◆ Windows XP, Windows 2000, Windows NT 4, and Windows 98

Provided that you have Microsoft Access installed on your computer, you can view the contents of any of these tables by simply double clicking from Windows Explorer on the file:

`c:\ILOG\OPLSt37\examples\opl\database\abridge.mdb`

Microsoft Access opens the database and displays the tables in alphabetical order. Double click on the table whose contents you want to see.



- ◆ UNIX

If you are working with Oracle on UNIX, and you followed the steps given for setting up the tables in your database, you can view the contents of any of these tables by issuing a `select` command at the prompt of the SQL shell:

`$ORACLE_HOME/bin/sqlplus`

The Task Table

The two-column table `Task` stores the names of the 46 tasks involved in the building of the bridge, together with their durations in days.

Each row of the table corresponds to a task, with the name of the task stored as a character string in the first column, and with its duration stored as an integer in the second column.

- ◆ Windows XP, Windows 2000, Windows NT 4, and Windows 98

Figure 10.1 shows a part of the table `Task` as you see it from Microsoft Access.

Name	Duration
a1	4
a2	2
a3	2
a4	2
a5	2
a6	5
ab1	1
ab2	1
ab3	1
ab4	1
ab5	1
ab6	1
b1	1
b2	1
b3	1
b4	1
b5	1
b6	1
k1	0
k2	0
l1	2
m1	16
m2	8
m3	8
m4	8

Figure 10.1 The Task Table

- ◆ UNIX

In order to see the contents of the table `Task`, launch the SQL shell then type:

```
SQL> select * from Task;
```

You see 46 rows selected.

## The Resource Table

The table `Resource` stores the different resources needed by the tasks during the construction of the bridge. Each row of the table corresponds to a resource and contains one column storing the name of the resource as a character string.

## The MAX and MIN Tables

The tables:

`MAX_EF`, `MAX_NF`, `MIN_AF`, `MIN_NF`, `MIN_SF`

store data describing the temporal constraints imposed on the different tasks involved in the project.

Each table contains three columns:

`Before`, `After`, `Distance`

The first two columns store the names of tasks, while the third column stores an elapsed number of days.

Depending on the table, each row is interpreted as a specific temporal constraint imposed on the start times and/or the end times of the tasks indicated.

For example, the data stored in the table `MAX_EF` corresponds to the constraint stating that the time between the completion of a particular formwork and the completion of its corresponding concrete foundation is at most four days.

- ◆ Windows XP, Windows 2000, Windows NT 4, and Windows 98

Figure 10.2 shows an example, table `MAX_EF`, as seen from Microsoft Access.

	Before	After	Distance
s1	b1		4
s2	b2		4
s3	b3		4
s4	b4		4
s5	b5		4
s6	b6		4
			0

**Figure 10.2** The `MAX_EF` Table

◆ UNIX

Here you can see the contents of the table `MAX_EF` as reported by the `select` command in the SQL shell:

```
SQL> select * from MAX_EF;
```

BEFORE	AFTER_	DIST
s1	b1	4
s2	b2	4
s3	b3	4
s4	b4	4
s5	b5	4
s6	b6	4

6 rows selected

**The Precedence Table**

The table `Precedence` indicates the precedence relationships that must exist between the different tasks of the project.

For each row, the task stored in the column `Before` must be completed before the beginning of the task stored in the column `After`.

**The Requirements Table**

Finally, the table `Requirements` stores the resources needed by different tasks. For each row, the first column `TaskName` indicates the name of a task, while the second column `ResName` contains the name of the particular resource needed by the task.

## The OPL Model

From the File menu in OPL Studio, select Open>Model, then select the appropriate model file.

- ◆ On Windows XP, Windows 2000, Windows NT 4, and Windows 98

```
c:\ILOG\OPLSt37\examples\opl\database\abridge.mod
```

- ◆ On UNIX

```
<installation-directory>/OPLSt37/examples/opl/database/obridge.mod
```

---

### Record Definitions

At the beginning of the model there are several definitions of OPL records, as shown in Code Sample 10.1.

```
struct TaskDuration {
    string task;
    int duration;
};

struct Distance {
    string before;
    string after;    //Task
    int dist;
};

struct Precedence {
    string before;    //Task
    string after;    //Task
};
.
.
.
struct TaskResource {
    string task;
    string resource;
};

struct Schedule {
    string task;
    int startTime;
    int endTime;
};
.
.
```

#### **Code Sample 10.1** *Record Definitions in the abridge.mod File*

These record definitions closely follow the actual structure of the rows in the different tables of the database. The OPL record TaskDuration is an example.

```
struct TaskDuration {
    string task;
    int duration;
};
```

This record corresponds to the two-column structure of the table Task. The field task corresponds to the column Name and the field duration corresponds to the column Duration in the table Task.

You will notice the type concordance between the table columns and the OPL fields. The column `Name` contains character strings, so the field `task` is of type `string`, the column `Duration` contains integers, so the field `duration` is of type `int`.

***Note:** Depending on your actual database system, it is possible that columns storing integer values need to be mapped to OPL fields of type `float`, rather than of type `int`. This is the case, for example, if you are using ODBC connected to an Excel database source, as the numeric values manipulated in Excel are of type `float`.*

Similarly, the OPL record `Distance` corresponds to the column structure of the tables:

`MAX_EF, MAX_NF, MIN_AF, MIN_NF, MIN_SF`

The OPL record `Precedence` corresponds to the column structure of the table `Precedence` and the OPL record `TaskResource` corresponds to the column structure of the table `Requirements`.

The OPL record `Schedule` corresponds to a new table, `Result`, that will be created at the end of the computation in order to store the results of the optimization.

---

## Connecting to the Database from OPL

Another interesting part of the model is the `DBconnection` statement you use to connect to the database.

### Connecting to ODBC

If you are using ODBC, the connection is established by the following statement:

```
DBconnection db("odbc", "abridge//");
```

The string passed as first argument indicates that you want to connect to a database source managed by ODBC. The string passed as second argument must respect the format below:

```
data source name/[user]/[password]
```

where `data source name` is the identifier you typed in the field `Data Source Name` when linking the database to ODBC (as explained in *Setting Up the Database* on page 217.). In our case, this identifier is `abridge`.

The fields `user` and `password` may be omitted, but note that the slash signs (/) are mandatory.

## Connecting to Oracle

**Note:** *If you are using an Oracle database, you should adapt the `DBconnection` statement to your particular case.*

The string passed as first argument must take the value `oraclex`, where `x` represents the particular version of the Oracle client you are using. A possible value is `oracle81`. The complete list of values can be found in the README file.

The string passed as second argument must respect the format below:

```
[user]/[password][@SQL Net id]
```

where `user` and `password` indicate the user name and the password that the database administrator has already assigned to you.

The field `SQL Net id` has the format:

```
<net:hostname> [:SID] for SQL Net V1
```

```
<instance name> for SQL Net V2
```

As an example, the distributed model file `obridge.mod` contains the connection statement:

```
DBconnection db("oracle8", "scott/tiger@ilog");
```

where the user `scott` with the password `tiger` will connect to the Oracle database called `ilog`.

---

## Reading From the Database

The rows contained in any table may be read into OPL by using the `DBread` statement. For example, with the following instruction you create an OPL set called `taskduration` which contains the rows of the table `Task`, each row becoming an OPL record of type `TaskDuration`:

```
{TaskDuration} taskduration from
  DBread(db, "select DURATION, NAME from Task")
  DBmapping {0->duration; 1->task;};
```

For the sake of the example, we wrote a `select` statement inverting the columns from the table, but note that the `DBmapping` part is not mandatory if there is a positional correspondence between the columns returned by the `select` statement and the fields of the OPL record.

Similarly, the OPL sets:

```
max_ef, max_nf, min_af, min_nf, min_sf, precedences
Resource, taskresource
```



are initialized with the rows of the corresponding tables from the database, as shown in Code Sample 10.2.

```
{Distance} max_ef from DBread(db, "select * from MAX_EF");
{Distance} max_nf from DBread(db, "select * from MAX_NF");
{Distance} min_af from DBread(db, "select * from MIN_AF");
{Distance} min_nf from DBread(db, "select * from MIN_NF");
{Distance} min_sf from DBread(db, "select * from MIN_SF");
{Precedence} precedences from DBread(db, "select * from PRECEDENCE");
{string} Resource from DBread(db, "select NAME from RESOURCE");
{TaskResource} taskresource from DBread(db, "select * from REQUIREMENTS");
```

#### **Code Sample 10.2** *Initializing OPL Sets*

Once these sets have been initialized, they can be used later in the model just like any other OPL set. For example, the instruction:

```
{string} Task = {t | <t,d> in taskduration};
```

collects the names of the tasks in a new set of strings called Task.

---

### **Creating a New Table and Updating the Database**

At the end of the optimization process, we want to store the optimal schedule obtained in a new database table.

First, we must collect the results of the optimization in a new OPL set. This set, called `resultSet`, will contain a record of type `Schedule` for each task involved in the bridge construction.

Each record stores the name of the task, and its start and end times, which were optimally computed by OPL:

```
{Schedule} resultSet = {#<task: t,
                        startTime: a[t].start,
                        endTime: a[t].start
                        + a[t].duration># |
                        t in Task};
```

By using the OPL statement `DBexecute`, you can create a new table, called `Result`, which has three columns, corresponding to the fields of the record `Schedule`.

If you are using ODBC with Microsoft Access, the instruction to create the new table is:

```
DBexecute(db, "create table Result (task string,
                                   startTime integer,
                                   endTime integer)");
```

If you are using Oracle, the instruction to create the new table is:

```
DBexecute(db, "create table Result (task VARCHAR(20),
                                   startTime NUMBER(6,0),
                                   endTime NUMBER(6,0))");
```

Finally, the members of the set `resultSet` can be inserted as rows in the table `Result` by using a `DBupdate` statement.

For ODBC, the insertion is made by the instruction:

```
DBupdate(db, "insert into Result (task, startTime, endTime)
           values (?, ?, ?)")(resultSet);
```


For Oracle, the insertion is made by the instruction:

```
DBupdate(db, "insert into Result (task, startTime, endTime)
           values (:1, :2, :3)")(resultSet);
```

The difference between the two `DBupdate` instructions is due to the different syntax for the placeholders inside the SQL request, imposed by the two database systems. In the case of ODBC you use a query sign as a placeholder, while in Oracle you use a column sign followed by a column number, with the columns numbered starting from one.

---

## Executing the Bridge Example

Click the Run button  in the tool bar of the OPL Studio Main window.

At the end of the execution you will see the following message in the Solutions notebook page:

Optimal Solution with Objective Value: 104

Also, OPL Studio will open the model browser. You can examine the model in the usual manner to see the contents of the various data structures in this example.

---

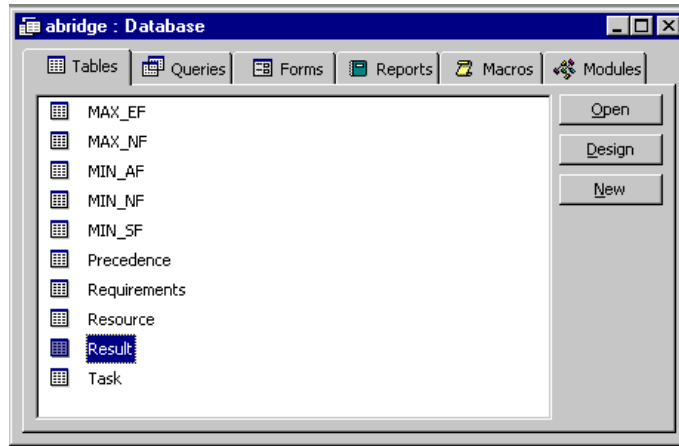
### Viewing the Result in the Database

When you have successfully executed the model from OPL Studio, you can view, in the database, the contents of the newly created table `Result`.

- ◆ Windows XP, Windows 2000, Windows NT 4, and Windows 98

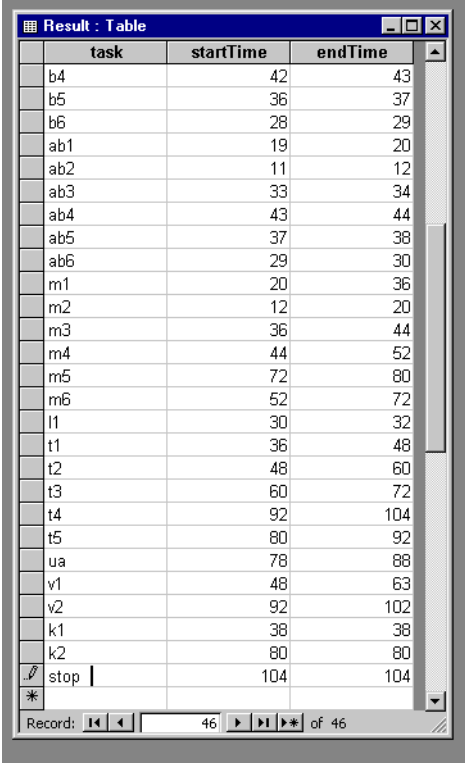
Close OPL Studio and Microsoft Access (if you are using it). Restart Microsoft Access by double clicking from Windows Explorer on the database `abridge.mdb`.

Figure 10.3 shows that the `Result` table has been added to the list of tables `abridge.mdb`.



*Figure 10.3 The Result Table*

Double click on `Result` in order to see the table's contents. The table contains 46 rows, each row corresponding to a task with its optimal start and end time, as you can see in Figure 10.4.



task	startTime	endTime
b4	42	43
b5	36	37
b6	28	29
ab1	19	20
ab2	11	12
ab3	33	34
ab4	43	44
ab5	37	38
ab6	29	30
m1	20	36
m2	12	20
m3	36	44
m4	44	52
m5	72	80
m6	52	72
l1	30	32
t1	36	48
t2	48	60
t3	60	72
t4	92	104
t5	80	92
ua	78	88
v1	48	63
v2	92	102
k1	38	38
k2	80	80
stop	104	104

**Figure 10.4** Contents of the Results Table

If you want to rerun the example, first remove the table `Result` from the database by selecting the name `Result` then pressing the Delete button on the keyboard.

#### ◆ UNIX

If you are working with Oracle on UNIX, you can view the contents of the table `Result` by typing the `select` command at the prompt of the SQL shell:

```
SQL> select * from Result;
```

You will see 46 rows selected.

If you want to rerun the example, remove the table `Result` from the Oracle database by typing a `drop` command at the prompt of the SQL shell:

```
SQL> drop table Result;
```

### Consulting the Result From Another Model

It is not mandatory to have Microsoft Access installed on your computer in order to view the contents of the table `Result`. As an alternative, you can type the following OPL instructions in a new model file:

```
DBconnection db("odbc", "abridge//"); //connect to the database

struct Schedule {
    string task;
    int startTime;
    int endTime;
};

{Schedule} see from
    DBread(db, "select * from Result"); //read the table Result

DBexecute(db, "drop table Result"); //remove the table from the database

solve;

display see; //display the result
```

#### **Code Sample 10.3** *OPL Instructions to View the Results Table*

Upon execution of this new model file, the set `see` will be displayed by OPL Studio in the Solution notebook page.

You can also examine the set `see` from the data structure tree built in the model browser.



## *Using OPLScript*

This chapter explains how to work with OPLScript, the OPL scripting language, in OPL Studio.

OPLScript enables you to:

- ◆ solve repeated instances of the same model
- ◆ make data modifications
- ◆ format output
- ◆ create algorithmic solutions where the output of one model is used as the input of a second model.

Refer to the *ILOG OPL Studio: Language Manual*, Chapters 16 and 17, for a detailed description of OPLScript.

---

## The Vellino Example

When using this example you will:

- ◆ open a script file and execute the script
- ◆ re-execute the script step by step in debug mode
- ◆ abort the execution while stepping in the script
- ◆ proceed with the execution while stepping in the script
- ◆ step out from a loop while stepping in the script

You do not use projects to manipulate scripts, you use script files with the extension `.osc`.

In this example you will be using the file `vellino.osc` from your release distribution. If you used the default directories at installation time, you can find this file in the following location:

- ◆ For UNIX systems

```
<installation-directory>/OPLSt37/examples/opl/scripts/vellino.osc
```

- ◆ For Windows XP, Windows 2000, Windows NT 4, and Windows 98

```
C:\ILOG\OPLSt37\examples\opl\scripts\vellino.osc
```

The application is a bin-packing configuration that, given a supply of components and bins of various types, must assign the components to the bins so that the bin constraints are satisfied and the smallest possible number of bins is used.

The application is described in detail in the *ILOG OPL Studio: Language Manual*, Section 16.4, Sequences of Models.

Here, we assume that you are familiar with this application and the solving strategy as it is explained in that document.


The file `vellino.osc` contains the script to implement the solving strategy. It invokes two models, contained in the files `genBin.mod` and `chooseBin.mod`.

Data for the model `genBin.mod` is contained in the file `genBin.dat`.

All these files are located in the same subdirectory as the script file `vellino.osc`.



## Opening the Script File

In the tool bar of the OPL Studio Main window, select File>Open>Script or, click on the icon .

ILOG OPL Studio displays a standard Open File dialog box for you to select the appropriate script file. Select `vellino.osc` from the `scripts` directory.

ILOG OPL Studio then displays the script in the editing area.

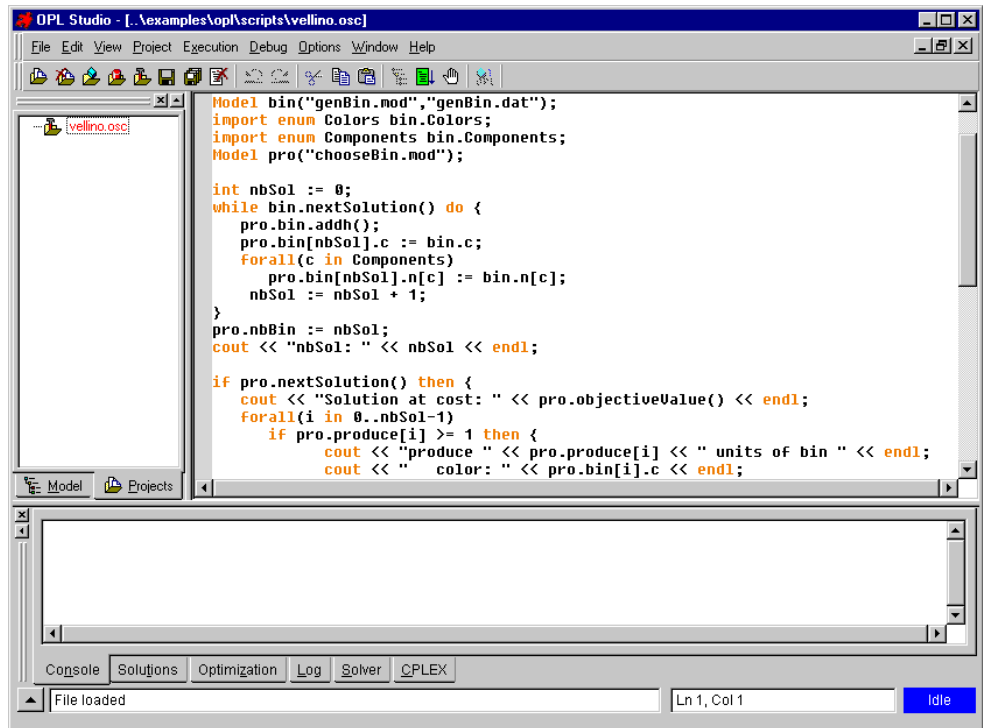



Figure 11.1 `vellino.osc`, OPL Script Example

## Executing the Script

In order to execute the loaded script, simply click the Run button  in the tool bar of the Main window.

**Note:** *If you happen to have a project open in the Main window, ILOG OPL Studio always gives precedence to the project file. The opened project file always gets executed. You will need to close the project before trying to execute the script file.*

Once the script is open, ILOG OPL Studio does the following:

- ◆ checks for syntactic or semantic errors
- ◆ executes the script, displaying the script file name in the status bar and changing the color patch to green
- ◆ if the script contains instructions of the form `cout << . . .`, the resulting output is printed in the Console notebook page
- ◆ upon completion, displays “OPL Studio is idle” in the status bar and changes the color patch to blue.

The Main window now looks like this:

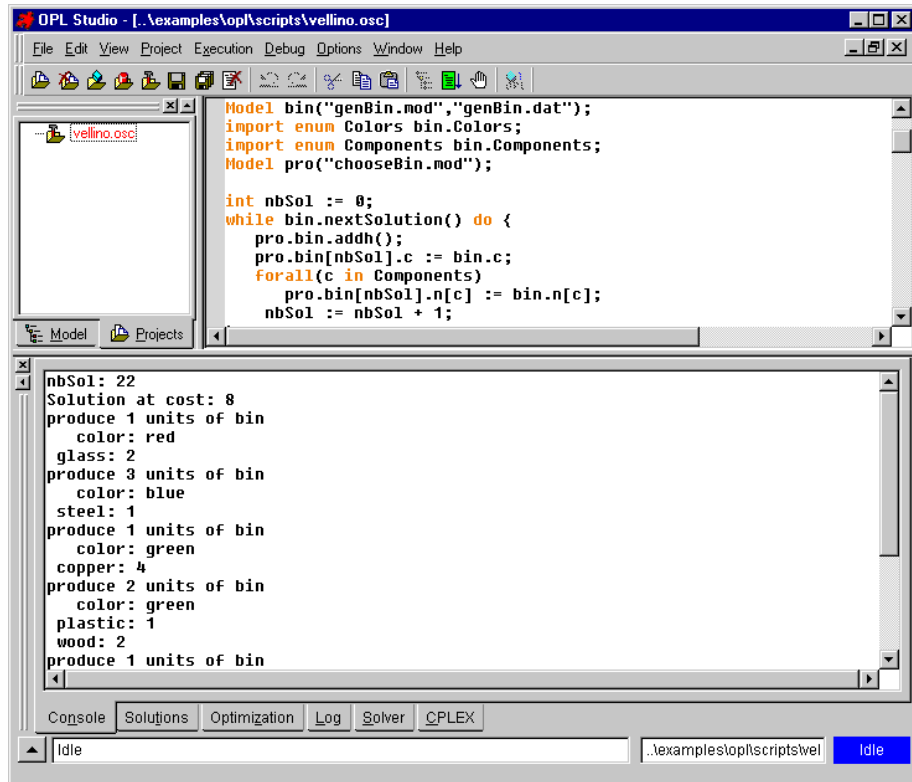


Figure 11.2 Execution of the vellino.osc Example

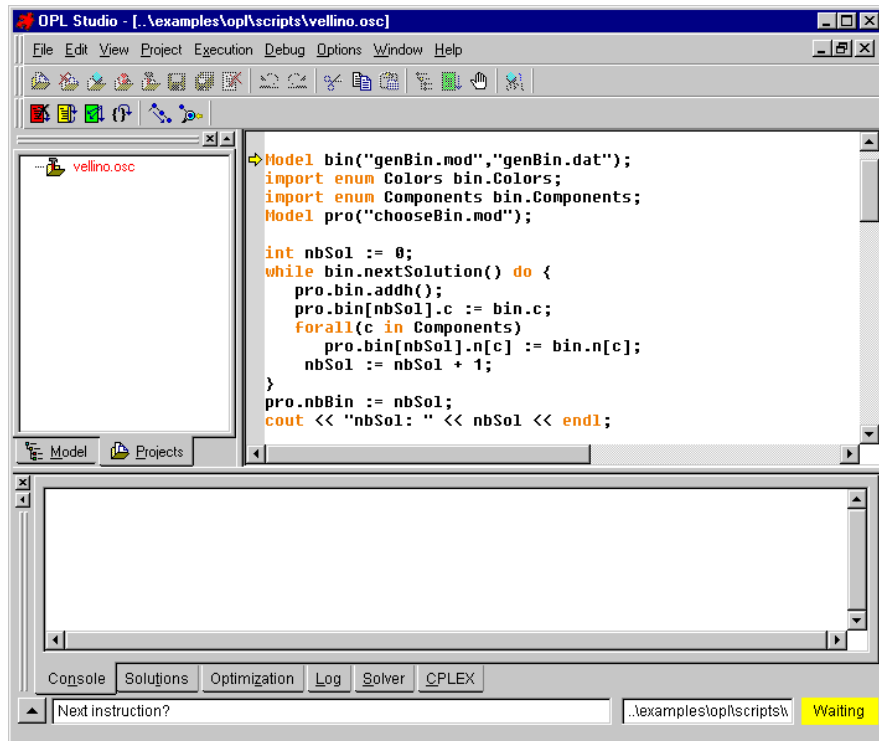
## Stepping in a Script

ILOG OPL Studio offers you the possibility of interactively executing a loaded script, step by step. This facility is useful for debugging a script.

### To Step to the First Instruction

Now that the script `vellino.osc` is open, select the option **Stepping in Script** from the **Debug** menu. Then click the **Run** button in order to start the execution of the script.


ILOG OPL Studio stops at the first instruction of the script, indicated by the current line arrow in the editor margin. At the same time, the status bar displays the message “Next instruction?” and the color patch changes to yellow and blinks (to stop the blinking, see [Blinking Status](#) on page 171).



**Figure 11.3** Step by Step Execution of a Script

---


### To Step to the Next Instruction

In order to execute the instruction indicated, click on the Next button  in the execution tool bar of the Main window.

ILOG OPL Studio executes this instruction then steps to the following instruction in the execution flow. By repeatedly clicking on the Next button, you can follow the execution of the script one instruction after another.

---

### To Abort the Execution

At any moment during the stepping in the script, you can abort the execution by clicking the Abort button  in the execution tool bar of the Main window.

In this case, you will get the message "OPLScript: Execution has aborted" in the Console notebook page. The status bar will indicate that OPL Studio is idle and the color patch will change to blue.

After aborting, you can relaunch the script in stepping mode, from the beginning, by clicking on Run.

---


### To Deselect the Stepping Option

If you want to run the script from the beginning without stepping, deselect the option Stepping in Script from the Debug menu, prior to clicking on Run.

As long as the option Stepping in Script is not deselected, the script will stop at the first instruction each time the Run button is clicked.

---

### To Continue without Stepping


At any moment during the stepping in the script, you can ask ILOG OPL Studio to continue until completion by clicking the Continue Run button  in the tool bar of the Main window.

OPL Studio will execute the rest of the script without stopping at instructions. The possible outputs of the script are printed in the Console notebook page. At the end of the execution, the status bar indicates that ILOG OPL Studio is idle.

You can relaunch the script in stepping mode, from the beginning, by clicking on Run.

---

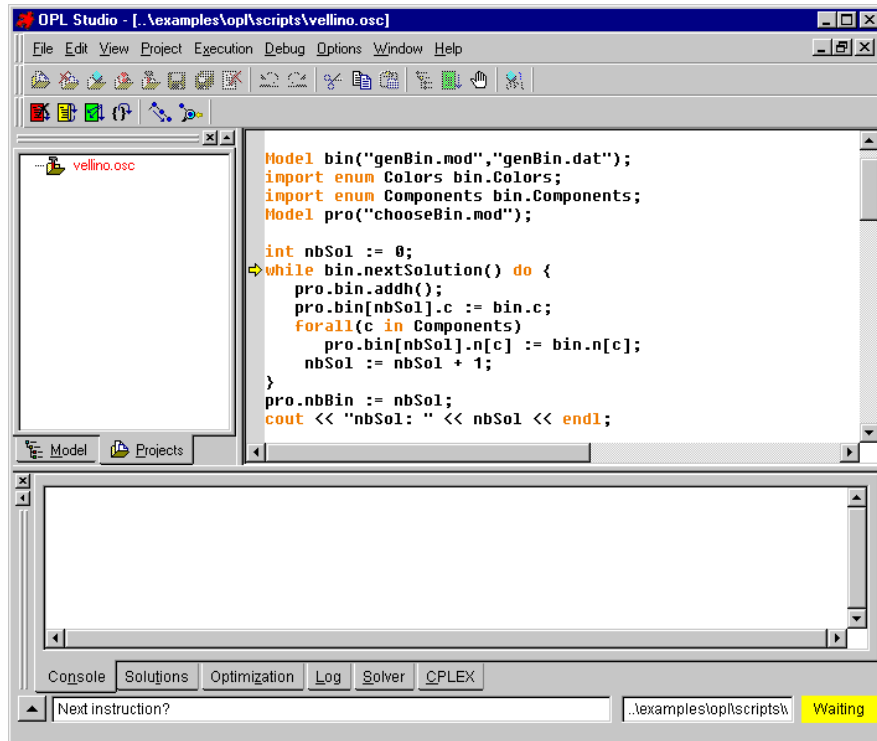
### To Step Out of a Loop

When you are stepping in a script and the current instruction is a looping instruction (i.e. `forall`, `while`, or `repeat`), you can ask ILOG OPL Studio to execute the loop without stopping at instructions, by clicking the Step Out button  in the tool bar.


OPL Studio will proceed with the entire loop, then it will stop at the first instruction after the loop in the execution flow.

For example, if you launched the script `vellino.osc` in stepping mode, you can first proceed by clicking the Next button several times until the current instruction, indicated by the yellow arrow in the editor margin (see Figure 11.4), becomes the `while` statement:

```
while bin.nextSolution() do {
```



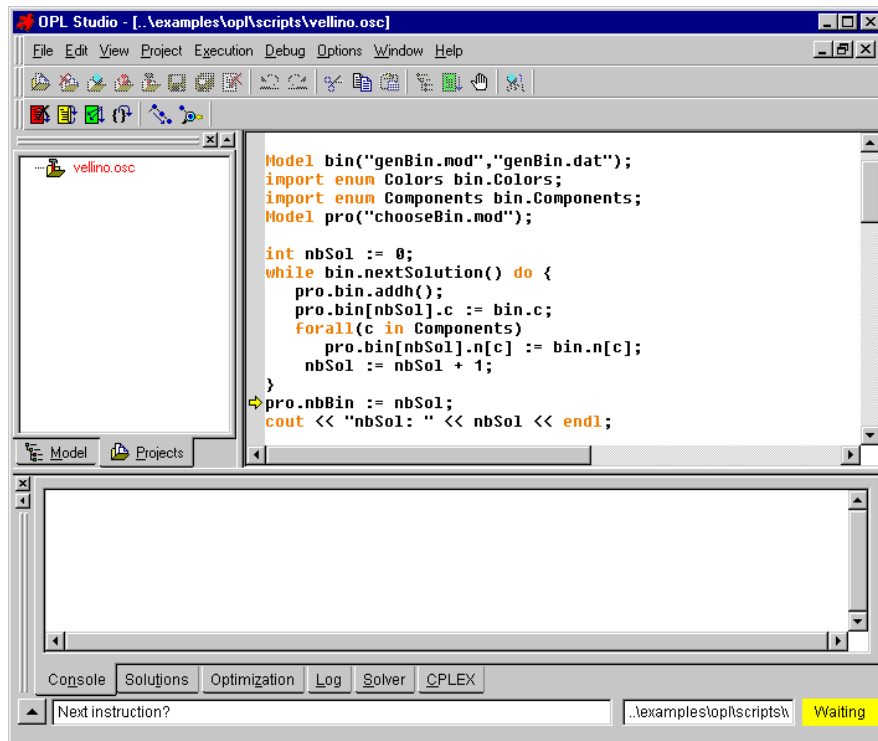
**Figure 11.4** A Loop in a Script

Now continue by clicking the Step Out button .

You will see that OPL Studio is not stepping inside the `while` loop and that the current instruction becomes:

```
pro.nbBin := nbSol;
```

which is the first instruction after the `while` loop.



**Figure 11.5** *Stepping Out of a Loop in OPLScript*

## To Complete the Execution

While stepping through a script, you can take ILOG OPL Studio to the idle state in one of three ways:

- ◆ click the Continue Run button
- ◆ click the Abort button
- ◆ click the Next button to follow the entire execution of the script, instruction by instruction. In this case, in order to avoid loops, you may use the Step Out button, as explained previously.

---

## Adding and Removing Breakpoints

You can place breakpoints against lines of code at which you want the execution to stop, either to display the value of a variable, or to continue the execution by clicking the Next button.

1. Uncheck the Stepping in Script option from the Debug menu. In the editor, click on the line:  

```
if pro.nextSolution()
```
2. Select the menu item Debug>Add/Remove Breakpoint. A red dot appears in the margin.
3. Run the script. It stops at the specified line.
4. By clicking on Next you can step in the subsequent lines of the script.

You can also continue the execution by clicking the Continue Run button.

---

## Closing the Script File

When you have completed the execution of a script, you can close the script file by selecting Close Current Editor from the File Menu.




## Generating Compiled Models

In this chapter you will learn how to generate compiled models that can be used with the OPL Component Libraries. Here, the basic steps are given for generating a compiled model. Examples and more detailed information are provided in the following documents:

*ILOG OPL Studio: Component Libraries User's Manual*

*ILOG OPL Studio: Component Libraries Reference Manual.*

1. Open a model in the usual way:
  - select File>Open>Model, or click on the corresponding button
  - select the model file from the standard Open File dialog box.Ensure that the focus is on the model.
2. Select the commands File>Generate Compiled Model File, or click on the corresponding button  in the tool bar.
3. The extension of the generated file is .opl. For example, if you generate a compiled model file for queens8.mod, the resulting file will be queens8.opl.
4. You can specify, in your default options settings, where you want these generated files to be placed (See *Setting Miscellaneous Options* on page 172.)

A compiled model is to be used with the OPLsolver API for compiled model files or buffers. OPLsolver is described in the *ILOG OPL Studio: Component Libraries Reference Manual*.



## OPL Parameters

The following table presents the OPL parameters in alphabetical order. The page numbers refer you to the chapter *Mathematical Programming*, where you will find a detailed description of each parameter.

**Table A.1** OPL Parameters

Parameter Name and Description	Type	Possible Values	Default	Page
aggCutLim Constraint aggregation limit for cut generation.	int	Any non-negative integer	3	203
aggFill Preprocessing aggregator fill.	int	Any non-negative integer	10	186
aggInd Preprocessing aggregator application limit.	int	-1 Automatic (1 for LP, infinite for MIP) 0 Do not use any aggregator — or — Any positive integer	-1	186

Parameter Name and Description	Type	Possible Values	Default	Page
barAlg Barrier algorithm.	int	0 Default primal-dual log barrier 1 Infeasibility-estimate start 2 Infeasibility-constant start 3 Standard barrier	0	205
barColNz Barrier column non-zeros.	int	0 Dynamically calculated — or — Any positive integer	0	207
barCrossAlg Barrier crossover method.	int	-1 No crossover 0 Automatic 1 Primal crossover 2 Dual crossover	0	206
barEpComp Barrier convergence tolerance.	float	Any positive number $\geq 1e-10$	1e-8	206
barGrowth Barrier growth limit.	float	Any positive number	1e12	207
barItLim Barrier iteration limit.	int	0 No barrier iterations — or — Any positive integer	2 100 000 000	208
barMaxCor Barrier maximum correction limit.	int	-1 Automatically determined 0 None Any positive integer	-1	208
barObjRng Barrier objective range.	float	Any positive number	1e20	207
barOOC Out-of-core barrier indicator.	int	0 Off 1 On	0	208
barOrder Barrier ordering algorithm.	int	1 Automatic 2 Approximate minimum degree (AMD) 3 Approximate minimum fill (AMF) 4 Nested dissection (ND)	4	206

Parameter Name and Description	Type	Possible Values	Default	Page
barStartAlg Barrier starting point algorithm.	int	1 Dual is 0 2 Estimate dual 3 Average of primal estimate, dual 0 4 Average of primal estimate, estimate dual	1	206
barThreads Barrier thread limit.	int	0 Determined by global thread default >0 Upper limit on threads for Parallel Barrier	0	208
basInterval Basis file saving frequency.	int	Any positive integer	2 100 000 000	183
BBInterval MIP strategy BBinterval.	int	0 Best estimate node always selected 1 Best bound node always selected — or — Any positive integer	7	190
bndStrenInd Bound strengthening indicator.	int	-1 Automatic 0 Off 1 On	-1	185
brDir MIP branching direction.	int	-1 Down branch selected first 0 Automatically determined 1 Up branch selected first	0	190
btTol MIP backtracking tolerance.	float	Any number between 0 and 1	0.9999	200
cliques MIP clique cuts indicator.	int	-1 Do not generate clique cuts 0 Automatically determined 1 Generate clique cuts moderately 2 Generate clique cuts aggressively	0	201
coeRedInd Coefficient reduction setting.	int	0 Do not use coefficient reduction 1 Reduce only to integral coefficients 2 Reduce all potential coefficients	2	186
covers MIP cover cuts indicator.	int	-1 Do not generate cover cuts 0 Automatically determined 1 Generate cover cuts moderately 2 Generate cover cuts aggressively	0	201
cplexLogFile Generates the CPLEX log file.	string	Empty by default, otherwise a string containing a file name.	Empty	177

Parameter Name and Description	Type	Possible Values	Default	Page
craInd  Simplex crash ordering.	int	Primal: 0 Ignore object coefficients during crash -1 or 1 Alternative ways of using objective coefficients Dual: 1 Default starting basis 0 or -1 Aggressive starting basis	1	180
cutLo  MIP lower cutoff tolerance.	float	Any number	-1e75	199
cutPass  Number of cutting plane passes.	int	-1 None 0 Automatically determined Any positive value, to indicate number of passes	0	196
cutsFactor  MIP row multiplier factor for cuts.	float	Any non-negative number	4.0	194
cutUp  MIP upper cutoff tolerance.	float	Any number	1e75	199
depInd  Dependency checker.	int	-1 Determined automatically 0 Off 1 On at beginning of preprocessing 2 On at end of preprocessing 3 On at beginning and end of preprocessing	-1	187
diveType  MIP dive strategy.	int	0 Automatically determined 1 Traditional dive 2 Probing dive 3 Guided dive	0	193
disjCuts  MIP disjunctive cuts indicator.	int	-1 Do not generate disjunctive cuts 0 Automatically determined 1 Generate disjunctive cuts moderately 2 Generate disjunctive cuts aggressively	0	203
dPriInd  Dual simplex pricing algorithm.	int	0 Determined automatically 1 Standard dual pricing 2 Steepest-edge pricing 3 Steepest-edge pricing in slack space 4 Steepest-edge pricing, unit initial norms 5 Devex pricing	0	181

Parameter Name and Description	Type	Possible Values	Default	Page
epAGap MIP absolute mipgap tolerance.	float	Any positive number	1e-6	198
epGap MIP relative mipgap tolerance.	float	Any number between 0.0 and 1.0	1e-04	198
epInt MIP integrality tolerance.	float	Any number between 1e-09 and 1.0	1e-05	199
epMrk Simplex Markowitz tolerance.	float	Any number between 0.0001 and 0.99999	0.01	183
epOpt Simplex optimality tolerance.	float	Any number between 1e-10 and 0.01	1e-06	183
epPer Simplex perturbation constant.	float	Any positive number $\geq 1e-8$	1e-6	184
epRHS Simplex feasibility tolerance.	float	Any number between 1e-10 and 0.01	1e-06	183
finalFactor Simplex final basis factorization after uncrush.	int	0 Off 1 On	1	181
flowCovers MIP flow cuts indicator.	int	-1 Do not generate flow cuts 0 Automatically determined 1 Generate flow cuts moderately 2 Generate flow cuts aggressively	0	202
flowPaths MIP flow path cut indicator.	int	-1 Do not generate flow path cuts 0 Automatically determined 1 Generate flow path cuts moderately 2 Generate flow path cuts aggressively	0	203
fracCand Candidate limit for generating Gomory fractional cuts.	int	Any non-negative integer	200	196

Parameter Name and Description	Type	Possible Values	Default	Page
<code>fracCuts</code> MIP Gomory fractional cuts indicator.	int	-1 Do not generate Gomory fractional cuts 0 Automatically determined 1 Generate Gomory fractional cuts moderately 2 Generate Gomory fractional cuts aggressively	0	203
<code>fracPass</code> Pass limit for generating Gomory fractional cuts.	int	Any non-negative integer	0	196
<code>GUBCovers</code> MIP GUB cuts indicator.	int	-1 Do not generate GUB cuts 0 Automatically determined 1 Generate GUB cuts moderately 2 Generate GUB cuts aggressively	0	202
<code>heurFreq</code> MIP heuristic frequency.	int	-1 None 0 Automatic — or — Any positive integer	0	190
<code>implBd</code> MIP implied bound cuts indicator.	int	-1 Do not generate implied bound cuts 0 Automatically determined 1 Generate implied bound cuts moderately 2 Generate implied bound cuts aggressively	0	202
<code>intSolLim</code> MIP solution limit.	int	Any positive integer	2 100 000 000	195
<code>itLim</code> Simplex maximum iteration limit.	int	Any non-negative integer	2 100 000 000	183
<code>MIPEmphasis</code> MIP emphasis indicator	int	0 Balance optimality and feasibility 1 Emphasize feasibility over optimality 2 Emphasize optimality over feasibility 3 Emphasize moving best bound 4 Emphasize hidden feasibles	0	178
<code>MIPOrdInd</code> MIP priority order indicator.	int	0 Off 1 On	1	192



Parameter Name and Description	Type	Possible Values	Default	Page
MIPOrdType  MIP priority order generation.	int	0 Do not generate a priority order 1 Use decreasing cost 2 Use increasing bound range 3 Use increasing cost per coefficient count	0	190
MIPThreads  MIP thread limit.	int	0 Determined by global thread default >0 Upper limit on threads for Parallel MIP	0	197
MIRCuts  MIP MIR (mixed integer rounding) cut indicator.	int	-1 Do not generate MIR cuts 0 Automatically determined 1 Generate MIR cuts moderately 2 Generate MIR cuts aggressively	0	204
netEpOpt  Network optimality tolerance.	float	Any number from 1e-4 to 1e-11	1e-6	210
netEpRHS  Network feasibility tolerance.	float	Any number from 1e-4 to 1e-11	1e-6	210
netFind  Simplex network extraction level.	int	1 Extract pure network only 2 Try reflection scaling 3 Try general scaling	2	210
netItLim  Network simplex iteration limit.	int	Any non-negative integer	2 100 000 000	210
netPPriInd  Network simplex pricing algorithm.	int	0 Automatic 1 Partial pricing 2 Multiple partial pricing 3 Multiple partial pricing with sorting	0	209
nodeFileInd  MIP node storage file indicator.	int	0 No node file 1 Node file in memory and compressed 2 Node file on disk 3 Node file on disk and compressed	1	197
nodeLim  MIP node limit.	int	Any non-negative integer	2 100 000 000	194

Parameter Name and Description	Type	Possible Values	Default	Page
nodeSel  MIP node selection strategy.	int	0 Depth-first search 1 Best-bound search 2 Best-estimate search 3 Alternate best-estimate search	1	191
objDif  MIP absolute objective difference cutoff.	float	Any number	0.0	199
objLLim  Simplex lower objective value limit.	float	Any number	-1e+75	184
objULim  Simplex upper objective value limit.	float	Any number	1e+75	184
perInd  Simplex perturbation indicator.	int	0 Off 1 On	0	184
perLim  Simplex perturbation limit.	int	0 Determined automatically — or — Any positive integer	0	182
PiecewiseCuts  Piecewise cuts indicator.	int	0 Off 1 On	1	179
pPriInd  Primal simplex pricing algorithm.	int	-1 Reduced-cost pricing 0 Hybrid reduced-cost & devex pricing 1 Devex pricing 2 Steepest-edge pricing 3 Steepest-edge pricing with slack initial norms 4 Full pricing	0	181
preCompress  Compression of model after presolve.	int	-1 Off 0 Automatic 1 On	0	186

Parameter Name and Description	Type	Possible Values	Default	Page
preDual Presolve dual setting.	int	-1 Off 0 Automatic 1 On	0	188
preInd Presolve indicator.	int	0 Off 1 On	1	187
prePass Presolve pass limit.	int	-1 Determined automatically 0 Do not use Presolve — or — Any positive integer	-1	187
preslNd Node presolve selector.	int	-1 No node presolve 0 Automatic 1 Force node presolve	0	192
priceLim Simplex pricing candidate list size.	int	0 Determined automatically — or — Any positive integer	0	182
probe MIP probe.	int	-1 No probing 0 Automatic 1 Probing level 1 2 Probing level 2 3 Probing level 3	0	192
reduce Primal and dual reduction type.	int	0 No primal and dual reductions 1 Only primal reductions 2 Only dual reductions 3 Both primal and dual reductions	3	188
reInv Simplex refactorization frequency.	int	0 Determined automatically — or — Any positive integer	0	182
relaxPreInd Relaxed LP presolve indicator.	int	0 Off 1 On	0	187
relObjDif MIP relative objective difference cutoff.	float	Any non-negative number	0.0	200

Parameter Name and Description	Type	Possible Values	Default	Page
RINSHeur MIP strategy RINS heuristic	int	-1 None 0 Automatic or, any positive integer	0	193
scaInd Scale parameter.	int	-1 No scaling 0 Equilibration scaling method 1 More aggressive scaling	0	181
singLim Simplex singularity repair limit.	int	Any non-negative integer	10	182
startAlg MIP starting LP algorithm.	int	0 Controlled by LP Method 1 Primal simplex 2 Dual simplex 3 Network simplex 4 Barrier	0	191
strongCandLim MIP candidate list.	int	Any positive integer	10	195
strongItLim MIP simplex iterations.	int	Any positive integer	0	195
strongThreadLim MIP parallel threads.	int	Any positive integer	1	195
subAlg MIP subproblem LP algorithm.	int	0 Automatic 1 Primal simplex 2 Dual simplex 3 Network optimizer followed by dual simplex 4 Barrier	0	191
subMIPNodeLim MIP subnode limit.	int	Any positive integer	500	197
symmetry Symmetry breaking cuts	int	0 Off 1 On	0	186

Parameter Name and Description	Type	Possible Values	Default	Page
<b>tiLim</b> Global time limit.	float	Any non-negative number	1e+75	176
<b>treLim</b> MIP tree memory limit.	float	Any positive number	128	196
<b>varSel</b> MIP variable selection strategy.	int	-1 Branch on variable with minimum infeasibility 0 Branch variable automatically selected 1 Branch on variable with maximum infeasibility 2 Branch based on pseudo reduced costs 3 Strong branching 4 Branch based on pseudo reduced costs	0	191
<b>workDir</b> The name of an existing directory in which CPLEX may store temporary working files.	string	Any valid string.  The default value is a dot (.), meaning the current local directory.	.	179
<b>workMem</b> Memory available, in megabytes, for working storage.	double	Any positive number	128	179



## List of Figures

1.1	Switching from the Japanese Version to the English (US) Version .....	21
1.2	ILOG OPL Studio Main Window .....	22
1.3	Overview of Commands in the Menu Bar .....	25
1.4	Handles on Dockable Windows .....	33
1.5	Title Bar on Floating Window (Windows XP, 2000, NT 4, 98).....	34
1.6	Handle on Floating UNIX Window (X-Windows).....	34
1.7	Selecting a New Model .....	36
1.8	New Model in Main Window .....	37
1.9	Simple OPL Statement in the <code>noname.mod</code> file .....	38
1.10	Error Message Displayed in Console Notebook Page .....	39
1.11	Text Editor after Hiding the Output and Model Windows .....	42
2.1	Project Window (Floating State) .....	54
2.2	Insert a Model File into a Project.....	55
2.3	Project Tree and Editing Area After Inserting a Model File.....	56
2.4	Inserting a Data File into a Project .....	57
2.5	Open Data File Dialog Box .....	57
2.6	Loading Data into a Project.....	58
2.7	Saving a Project .....	59
2.8	Project Tree for <code>product.prj</code> Example .....	59
2.9	Project Options Notebook.....	60
2.10	Model Browser for <code>product.prj</code> Example .....	62
2.11	"Next solution?" Message .....	63
2.12	Solutions Notebook Page for <code>product.prj</code> Example.....	64

2.13	Log Notebook Page for <code>product.prj</code> Example .....	64
2.14	Solver Notebook Page for <code>product.prj</code> Example .....	65
2.15	CPLEX Notebook Page for <code>product.prj</code> Example .....	66
2.16	Model Browser for <code>product.mod</code> File .....	67
2.17	Open View Button .....	68
2.18	Main Window After Executing <code>product.prj</code> Example .....	72
2.19	Project Tree in the Work Space .....	74
2.20	Setting an Active Project .....	75
3.1	Loading a Data File for the Car Sequencing Example .....	83
3.2	Main Window for the Car Sequencing Example .....	84
3.3	<code>slot</code> Variable Results for Solution[1] .....	86
3.4	Alternative View of <code>slot</code> Variable Results for Solution[1] .....	86
3.5	<code>setup</code> Variable Results for Solution[1] .....	87
3.6	Alternative View of <code>setup</code> Variable Results for Solution[1] .....	87
3.7	Model Browser for <code>car.prj</code> (Floating State) .....	89
3.8	Selecting a Dynamic Display Option .....	91
3.9	Setup Notebook Page Showing Boolean Display .....	92
4.1	Main Window for the House Building Example .....	96
4.2	Model Browser for <code>house2.mod</code> (Floating State) .....	97
4.3	Main Window with <code>house2.mod</code> Solution .....	99
4.4	Activities Results for <code>house2.mod</code> Example .....	100
4.5	Alternative View of Activities Results for <code>house2.mod</code> Example .....	101
4.6	Gantt Chart of Activities Results for <code>house2.mod</code> Example .....	102
4.7	Gantt Chart Split into Four Views .....	104
4.8	Activities Notebook Page for <code>house2.mod</code> Example .....	105
4.9	Sorted Activities Gantt Chart for <code>house2.mod</code> Example .....	106
4.10	Discrete Resource Results for <code>house2.mod</code> Example .....	107
4.11	Alternative View of Discrete Resources Results for <code>house2.mod</code> Example .....	108
4.12	Optimization Notebook Page for <code>house2.mod</code> Example .....	109
4.13	Solver Notebook Page for <code>house2.mod</code> Example .....	109
4.14	Closing a Model in the Projects Page .....	111
4.15	Closing a Model in the Editing Area .....	111
7.1	The Eight Queens Model with the Default Search Strategy .....	128
7.2	Project Tree for the Frequency Allocation Example (in Dockable Window) .....	133
7.3	Main Window for the Frequency Allocation Example .....	134



<b>7.4</b>	Browsing an Active Model - Frequency Allocation Example .....	<b>135</b>
<b>7.5</b>	Model Browser for the Frequency Allocation Example .....	<b>135</b>
<b>7.6</b>	Debug Menu .....	<b>136</b>
<b>7.7</b>	Main Window in Debug Mode with Stop at Choice Point Option .....	<b>138</b>
<b>7.8</b>	Stack Window (Floating) at the First Choice Point .....	<b>139</b>
<b>7.9</b>	Inspector Window (Floating) at the First Choice Point .....	<b>139</b>
<b>7.10</b>	Stack Window at Second Choice Point. ....	<b>140</b>
<b>7.11</b>	Inspector Window at Second Choice Point. ....	<b>140</b>
<b>7.12</b>	Stack Window at the Fourth Choice Point. ....	<b>141</b>
<b>7.13</b>	Inspector Window at the Fourth Choice Point. ....	<b>141</b>
<b>7.14</b>	Main Window in Debug Mode with Stepping Option .....	<b>143</b>
<b>7.15</b>	Stack and Inspector Windows at First Instruction of Stepping Option .....	<b>144</b>
<b>7.16</b>	Stack and Inspector Windows at Second Instruction of Stepping Option .....	<b>145</b>
<b>7.17</b>	Stack and Inspector Windows at Sixth Instruction of Stepping Option .....	<b>146</b>
<b>7.18</b>	Order of the Search Tree Exploration with the Default Depth-First Search .....	<b>147</b>
<b>7.19</b>	Part of the Search-tree Exploration Order with the SBS .....	<b>151</b>
<b>7.20</b>	Main Window After Leaving Debug Mode. ....	<b>152</b>
<b>7.21</b>	Search Tree and Objective Bounds, after One Choice using Solver without Linear Relaxation. ....	<b>155</b>
<b>7.22</b>	Search Tree without Linear Relaxation after 14 <code>Next</code> Commands. ....	<b>156</b>
<b>7.23</b>	Search Tree after One Choice using Solver with Linear Relaxation at Each Node .....	<b>157</b>
<b>7.24</b>	Search Tree with Linear Relaxation after 14 <code>Next</code> commands. ....	<b>159</b>
<b>7.25</b>	Exploring the Search Tree in the Slice-Based Search .....	<b>161</b>
<b>8.1</b>	Default Options - Constraint Programming .....	<b>165</b>
<b>8.2</b>	Editor Options .....	<b>166</b>
<b>8.3</b>	Font Chooser Dialog Box .....	<b>167</b>
<b>8.4</b>	Color Chooser Dialog Boxes. ....	<b>168</b>
<b>8.5</b>	Output Options .....	<b>169</b>
<b>8.6</b>	Advanced Options. ....	<b>171</b>
<b>8.7</b>	Miscellaneous Options .....	<b>172</b>
<b>10.1</b>	The <code>Task</code> Table .....	<b>219</b>
<b>10.2</b>	The <code>MAX_EF</code> Table .....	<b>220</b>
<b>10.3</b>	The <code>Result</code> Table .....	<b>227</b>
<b>10.4</b>	Contents of the <code>Results</code> Table .....	<b>228</b>
<b>11.1</b>	<code>vellino.osc</code> , OPL Script Example .....	<b>233</b>
<b>11.2</b>	Execution of the <code>vellino.osc</code> Example. ....	<b>235</b>

<b>11.3</b>	Step by Step Execution of a Script .....	<b>236</b>
<b>11.4</b>	A Loop in a Script .....	<b>238</b>
<b>11.5</b>	Stepping Out of a Loop in OPLScript .....	<b>239</b>





## ***List of Code Samples***

<b>1.1</b>	Example of an OPL Statement .....	<b>37</b>
<b>2.1</b>	OPL Model for the Production Planning Example ( <code>product.mod</code> ) .....	<b>52</b>
<b>2.2</b>	OPL Data for the Production Planning Example ( <code>product.dat</code> ) .....	<b>53</b>
<b>2.3</b>	OPL Named Data for Production Planning Example ( <code>productn.dat</code> ).....	<b>73</b>
<b>3.1</b>	OPL Model for the Car Sequencing Example ( <code>car.mod</code> ).....	<b>81</b>
<b>3.2</b>	OPL Data for the Car Sequencing Example ( <code>car.dat</code> ).....	<b>82</b>
<b>4.1</b>	OPL Model for the House Building Example ( <code>house2.mod</code> ) .....	<b>95</b>
<b>7.1</b>	OPL Model for the Frequency Allocation Example with Search Procedure .....	<b>131</b>
<b>7.2</b>	OPL Data for the Frequency Allocation Example ( <code>alloc.dat</code> ).....	<b>132</b>
<b>10.1</b>	Record Definitions in the <code>abridge.mod</code> File .....	<b>222</b>
<b>10.2</b>	Initializing OPL Sets .....	<b>225</b>
<b>10.3</b>	OPL Instructions to View the <code>Results</code> Table.....	<b>229</b>



# Index

## A

Abort button **32, 40, 71**  
 Abort command **28**  
 active model **76**  
 active project **74**  
 active script **76**  
 Activities notebook page **105**  
 activity domains after propagation **116**  
 Activity Domains window **114**  
 activity domains, color options **171**  
 activity, definition **94**  
 Add New Data File command **28**  
 Add New Model File command **28**  
 Add/Remove Breakpoint button **32**  
 adding  
     data files to projects **57, 58**  
     model files to projects **55, 58**  
 advanced options **171**  
 aggCutLim parameter **203, 243**  
 aggFill parameter **186, 243**  
 aggInd parameter **186, 243**  
 algorithm animation **116**  
 algorithms used for solving **65**  
 All Solutions command **28**  
 alternative data files **73**

## B

backtrackable 2D graphics **119**

barAlg parameter **205, 244**  
 barColNz parameter **207, 244**  
 barCrossAlg parameter **206, 244**  
 barEpComp parameter **206, 244**  
 barGrowth parameter **207, 244**  
 barItLim parameter **208, 244**  
 barMaxCor parameter **208, 244**  
 barObjRng parameter **207, 244**  
 barOOC parameter **244**  
 barOOC parameter **208**  
 barOrder parameter **206, 244**  
 barStartAlg parameter **206, 245**  
 barThreads parameter **208, 245**  
 basic concepts  
     checking for syntactic errors **39**  
     creating files **36**  
     executing models or projects **38**  
     opening files **36**  
     processing directives **40**  
     terminating ILOG OPL Studio **40**  
 basInterval parameter **183, 245**  
 batch mode **18**  
 BBInterval parameter **190, 245**  
 bndStrenInd parameter **185, 245**  
 Board keyword **119**  
 brDir parameter **190, 245**  
 breakpoints **240**  
 Browse Active Model command **28**  
 Browse Model command **28**  
 btTol parameter **245**

building the data structure **89, 96**

buttons

- Abort **32, 40, 71**
- Add/Remove Breakpoint **32**
- Close Current Editor **31**
- Continue Run **32, 40, 71**
- Copy **31**
- Cut **31**
- Generate Compiled Model File **32**
- Inspect Current Node **32**
- Load Data File **31**
- Load Model File **31**
- Load Project File **31**
- Load Script File **31**
- Next **32, 40, 71**
- Paste **31**
- Rebuild Browser Information **32**
- Redo **31**
- Run **32**
- Save All Files **31**
- Save and Close Project **31**
- Save Editor Content **31**
- Step Out **32**
- Undo **31**
- View Choice Stack **32**

## C

car sequencing tutorial

- car.dat file **82**
- car.mod file **80**
- car.prj file **83**
- closing the project file **92**
- continuing the execution **88**
- displaying variable results **91**
- dynamic display **90**
- examining the solution **86**
- executing the project **85**
- looking at the model structure **89**
- Main window **84**
- opening a project file **83**
- problem description **80**

Cascade command **29**

changing colors **168**

changing fonts **167**

checking for errors **39**

choice point **29, 32, 66, 128, 136, 139, 142, 146, 154, 160**

choice point, in bridgebr example **116**

choice stack **124**

Choice Stack command **28**

cliques parameter **201, 245**

Close Current Editor button **31**

Close Current Editor command **26**

Close Project command **28**

CloseActive Project command **26**

closing models **111**

closing projects **74, 92**

closing the active file **31**

coeRedInd parameter **186, 245**

Color Chooser dialog box **168**

color patch, setting the blink option **171**

colors

changing **168**

comments **168**

editor **168**

error **168**

label **169**

OPL keywords and functions **168**

OPLScript **168**

options **169**

commands

Debug

Add/Remove Break Point **29**

Display Search Tree **29**

Stepping in Model **29**

Stepping in Script **29**

Stop at Choice Point **29**

Stop at Solution **29**

Edit

Comment **27**

Complete Word **27**

Copy **27**

Cut **27**

Find **27**

Find Next **27**

Find Previous **27**

Go To **27**

Indent Lines **27**

Kill Line **27**

Outdent Lines **27**



- Paste **27**
- Recenter **27**
- Redo **26**
- Replace **27**
- Select All **27**
- Uncomment **27**
- Undo **26**
- Execution
  - Abort **28**
  - All Solutions **28**
  - Browse Active Model **28**
  - Continue Run **29**
  - Next **29**
  - Run **28**
  - Step Out **29**
- File
  - Close Active Project **26**
  - Close Current Editor **26**
  - Dump Active Model and **26**
  - Generate Compiled Model File **26**
  - New **26**
  - Open **26**
  - Quit **26**
  - Recent Files **26**
  - Save **26**
  - Save All **26**
  - Save As **26**
- Help
  - Contents **30**
  - Keyword Help **30**
- Options
  - Customize Active Project Options **29, 164**
  - Customize Default Options **29, 164**
- Project
  - Add New Data File **28**
  - Add New Model File **28**
  - Browse Model **28**
  - Close Project **28**
  - Insert Existing Data File **28**
  - Insert Existing Model File **28**
  - Project Options **28**
  - Save the Project **28**
  - Save the Project As **28**
  - Set as Active Project **28**
- View
  - Choice Stack **28**
  - Inspect Current Node **28**
  - Output **28**
  - Workspace **27**
- Window
  - Cascade **29**
  - Tile Horizontally **29**
  - Tile Vertically **29**
- comment colors **168**
- Comment command **27**
- compiled model files **26**
- compiled models **173**
- Complete Word command **27**
- configuration file `oplst3.config` **18**
- Console notebook page **40**
- constraint programming options **165**
- Continue Run button **32, 40, 71**
- Continue Run command **29**
- continuing executions
  - car sequencing **88**
  - production planning **71**
- Copy button **31**
- Copy command **27**
- `covers` parameter **201, 245**
- CPLEX notebook page **211**
- `cplexLogFile` parameter **177, 245**
- `craInd` parameter **180, 246**
- creating files **36**
- creating projects **54, 59**
- current node inspector **124**
- Customize Active Project Options command **29**
- Customize Default Options command **29**
- customizing default options **164**
- customizing OPL Studio
  - changing colors **168**
  - changing fonts **167**
  - constraint programming options **165**
  - setting advanced options **171**
  - setting output options **169**
- customizing project options **60, 61, 164**
- Cut button **31**
- Cut command **27**
- `cutLo` parameter **199, 246**
- `cutPass` parameter **196, 246**

`cutsFactor` parameter **194, 246**  
`cutUp` parameter **199, 246**

## D

data files **58, 83**  
 data files, description **35**  
 data initialization **73**  
 data structure information **89, 96**  
 data, loading into editor **133**  
 database connection  
   bridge example **213, 216, 226**  
   installing the database **217**  
   ODBC **223, 225**  
   Oracle **224, 225**  
   reading from a database **224**  
   storing results in a database **225**  
   the OPL model **222**  
   updating a database **225**  
   viewing the data tables **218**  
 database connectivity **214**  
 databases supported in OPL **214**  
 DBconnection OPL instruction **214**  
 Debug menu  
   Add/Remove Break Point **29**  
   Display Search Tree **29**  
   Stepping in Model **29**  
   Stepping in Script **29**  
   Stop at Choice Point **29**  
   Stop at Solution **29**  
 debugging OPL models  
   executing in debug mode **137, 142**  
   Inspector window **139**  
   options **136**  
   processing directives **141**  
   setting options **136**  
   Stack window **139**  
   Stepping in Model option **142**  
   Stop at Choice Point **136**  
 debugging OPL scripts  
   stepping in a script **236**  
   stepping out from a loop **237**  
 Default Options dialog box **164**  
 depInd parameter **187, 246**

dialog boxes  
   Color Chooser **168**  
   Default Options **164**  
   Font Chooser **167**  
   Project Options **60**  
 disjCuts parameter **203, 246**  
 displaying  
   activities results **100, 101, 105**  
   Inspector window **139, 144**  
   model data structure **89, 96**  
   results **90**  
   sorted activities **106**  
   Stack window **139, 144**  
   variable arrays **87**  
 diveType parameter **193, 246**  
 dockable elements **33**  
 docking a window **34**  
 dPriInd parameter **181, 246**  
 drawing board  
   docking, undocking **119, 121**  
   setting the docking option **171, 172**  
   the Euler example **122**  
   the map example **122**  
   the square example **120**  
 Dump Active Model and Result command **26**  
 dynamic display **90, 113**

## E

Edit menu  
   Comment **27**  
   Complete Word **27**  
   Copy **27**  
   Cut **27**  
   Find **27**  
   Find Next **27**  
   Find Previous **27**  
   Go To **27**  
   Indent Lines **27**  
   Kill Line **27**  
   Outdent Lines **27**  
   Paste **27**  
   Recenter **27**  
   Redo **26**  
   Replace **27**

- Select All **27**
  - Uncomment **27**
  - Undo **26**
  - editing area, description **23**
  - editing mode **37**
  - editor
    - customizing **46**
    - Emacs mode **43**
    - how to use **41**
    - loading data **133**
    - Multi-Document Interface **41**
    - quick reference **43**
    - visual mode **43, 46**
  - editor colors **168**
  - epAGap parameter **198, 247**
  - epGap parameter **198, 247**
  - epInt parameter **199, 247**
  - epMrk parameter **183, 247**
  - epOpt parameter **183, 247**
  - epPer parameter **184, 247**
  - epRHS parameter **183, 247**
  - error checking
    - description of **39, 62**
    - display colors **168**
    - error message format **40**
  - Euler example **122**
  - examining
    - activities results **100, 101, 105**
    - model solutions **63, 88, 100**
    - variable results **86, 87**
  - examples
    - abridge.mod (using a DB) **216, 226**
    - bridgebr.prj (activities domain) **116**
    - car sequencing **80**
    - Euler (drawing board) **122**
    - frequency allocation **130**
    - house building **94**
    - map (drawing board) **122**
    - production planning **52**
    - square (drawing board) **120**
    - Vellino (OPLScript) **232**
  - Excel, paste results to **88**
  - executing
    - in debug mode **137**
    - models **38, 98**
    - next command **140**
    - projects **38, 62, 85, 137, 142**
    - scripts **234**
    - termination **152**
  - execution events, description **62, 85, 98**
  - Execution menu
    - Abort **28**
    - All Solutions **28**
    - Browse Active Model **28**
    - Continue Run **29**
    - Next **29**
    - Run **28**
    - Step Out **29**
  - execution steps **71**
  - execution tool bar buttons **32, 71**
  - execution tool bar, description **23**
  - exported file formats **177**
- ## F
- file export **177**
  - file formats
    - LP **177**
    - MPS **177**
    - REW **177**
    - RLP **177**
    - SAV **177**
  - File menu
    - Close Active Project **26**
    - Close Current Editor **26**
    - Dump Active Model and **26**
    - Generate Compiled Model File **26**
    - New **26**
    - Open **26**
    - Quit **26**
    - Recent Files **26**
    - Save **26**
    - Save All **26**
    - Save As **26**
  - file naming conventions **35**
  - file saving preferences **47**
  - file types in OPL
    - data file **35**
    - model **35**
    - project **35**
    - script **35**

files referenced in OPL/OPLScript **172**  
 finalFactor parameter **181, 247**  
 Find command **27**  
 Find Next command **27**  
 Find Previous command **27**  
 floating a window **33**  
 flowCovers parameter **202, 247**  
 flowPaths parameter **203, 247**  
 Font Chooser dialog box **167**  
 fonts  
     changing **167**  
     options **169**  
 fracCand parameter **196, 247**  
 fracCuts parameter **203, 248**  
 fracPass parameter **196, 248**  
 frequency allocation **140**  
 frequency allocation tutorial  
     alloc.dat file **132**  
     alloc.mod file **130**  
     alloc.prj file **133**  
     continuing the execution **140**  
     executing the project **137**  
     looking at the model structure **135**  
     Main window **134**  
     model browser **135**  
     problem description **130**  
     processing directives **141**  
     setting the debug option **136**  
     Stack and Inspector windows **139**

## G

Gantt chart  
     how to use **102**  
     showing activities results **101**  
 Go To command **27**  
 GUBCovers parameter **248**  
 gubCovers parameter **202**

## H

Help menu  
     Contents **30**  
     Keyword Help **30**

help on keywords  
     UNIX platforms **49**  
     Windows platforms **48**  
 heurFreq parameter **190, 248**  
 house building tutorial  
     Activities notebook page **105**  
     activities results **100**  
     closing the model file **111**  
     completing the execution **110**  
     discrete resources bar chart **108**  
     examining the solution **100**  
     executing the model **98**  
     house2.mod file **95**  
     list of activities **94**  
     looking at the model structure **96**  
     Main window **96**  
     Main window after execution **99**  
     Optimization notebook page **109**  
     problem description **94**  
     Resources results **107**  
     Solutions notebook page **107**  
     Solver notebook page **109**  
     Sorted Activities notebook page **106**

## I

implBd parameter **202, 248**  
 include files **172**  
 Indent Lines command **27**  
 initialization of data **73**  
 Insert Existing Data File command **28**  
 Insert Existing Model File command **28**  
 Inspect Current Node button **32**  
 Inspect Current Node command **28**  
 Inspector window **139, 144**  
 installation directory **51, 80, 93, 123**  
 intSolLim parameter **195, 248**  
 itLim parameter **183, 248**

## J

Japanese version  
     database restriction **21, 213**  
     switching to English version **21**

**K**

keyword help **48**

Kill Line command **27**

**L**

label colors **169**

launching ILOG OPL Studio **18**

Load Data File button **31**

Load Model File button **31**

Load Project File button **31**

Load Script File button **31**

loading data into the editor **58, 83, 133**

localization

    database restriction in Japanese version **21, 213**

    switching from Japanese to English **21**

log notebook page **64**

LP file format **177**

**M**

Main window

    button descriptions **31**

    command descriptions **26**

    description **22**

    editing area **23**

    execution tool bar **23**

    execution tool bar buttons **32**

    line number, column number **23**

    menu bar **23**

    model browser **23**

    output notebook **23**

    pathname **23**

    project tree **23**

    status bar **23**

    status box **24**

    tool bar **23**

    work space **23**

MDI (Multi-Document Interface) **41**

menu bar description **23**

MIPeMphasis parameter **178, 248**

MIPOrdInd parameter **192, 248**

MIPOrdType parameter **190, 249**

MIPThreads parameter **197, 249**

MIRCuts parameter **204, 249**

mode selectors for the Activity Domains window **115**

model browser

    building the data structure **89, 96**

    collapsed structure **72**

    description **23, 67**

    displaying a Gantt chart **101**

    displaying problem results **70**

    for alloc.prj **135**

    for car.prj **89**

    for house2.mod **97**

    for product.prj **62**

    navigating the model file **68**

    using dynamic display **90**

model structure

    rebuilding **89**

models

    closing **111**

    description **35**

    executing **98**

    opening **95**

    setting as active **76**

modifying

    colors **169**

    fonts **169**

MPS file format **177**

Multi-Document Interface **41**

**N**

named initialization of records **73**

netEpOpt parameter **210, 249**

netEpRHS parameter **210, 249**

netFind parameter **210, 249**

netItLim parameter **210, 249**

netPPriInd parameter **209, 249**

New command **26**

Next button **32, 40, 71**

Next command **29**

node inspector **124**

nodeFileInd parameter **197, 249**

nodeLim parameter **194, 249**

nodeSel parameter **191, 250**

## notebook pages

- Activities **105**
- Console **40**
- CPLEX **211**
- Optimization **64, 109**
- setup **91**
- Solutions **63**
- Solver **65, 109**
- Sorted Activities **106**

**O**

- objDif parameter **199, 250**
- objLLim parameter **184, 250**
- objULim parameter **184, 250**
- ODBC **223, 225**
- online help **48**
- Open command **26**
- opening
  - model files **95**
  - projects **83**
  - scripts **233**
- opening files **36**
- OPL keyword and function colors **168**
- OPL Studio options
  - export file format **177**
  - refresh rate **177**
- OPLScript
  - debugging **236**
  - description **232**
  - example **232**
  - executing **234**
  - executing step by step **236**
  - keyword colors **168**
  - opening a script file **233**
  - stepping out from a loop **237**
- OPLScript, features **231**
- oplst3.config file **18**
- Optimization notebook page **64, 109**
- optimization statement **99, 109**
- options
  - activities **170**
  - advanced **171**
  - console font **170**
  - constraint programming **165**

output **169**

- pathname for compiled models **173**
- precision **170**
- project **60, 61**
- sorted activities **170**
- textual result **170**

## Options menu

- Customize Active Project Options **29, 164**
- Customize Default Options **29, 164**

options, setting **75**Oracle **224, 225**Outdent Lines command **27**Output command **28**output notebook, description **23**

## output options

- hide zeros **169**
- setting **169**
- square matrix **169**

**P**

## parameters

- aggCutLim **203, 243**
- aggFill **186, 243**
- aggInd **186, 243**
- barAlg **205, 244**
- barColNz **207, 244**
- barCrossAlg **206, 244**
- barEpComp **206, 244**
- barGrowth **207, 244**
- barItLim **208, 244**
- barMaxCor **208, 244**
- barObjRng **207, 244**
- barOOC **208, 244**
- barOrder **206, 244**
- barStartAlg **206, 245**
- barThreads **208, 245**
- basInterval **183, 245**
- BBInterval **190, 245**
- bndStrenInd **185, 245**
- brDir **190, 245**
- btTol **200, 245**
- cliques **201, 245**
- coeRedInd **186, 245**
- covers **201, 245**

cplexLogFile **177, 245**  
 craInd **180, 246**  
 cutLo **199, 246**  
 cutPass **196, 246**  
 cutsFactor **194, 246**  
 cutUp **199, 246**  
 depInd **187, 246**  
 disjCuts **203, 246**  
 diveType **193, 246**  
 dPriInd **181, 246**  
 epAGap **198, 247**  
 epGap **198, 247**  
 epInt **199, 247**  
 epMrk **183, 247**  
 epOpt **183, 247**  
 epPer **184, 247**  
 epRHS **183, 247**  
 finalFactor **181, 247**  
 flowCovers **202, 247**  
 flowPaths **203, 247**  
 fracCand **196, 247**  
 fracCuts **203, 248**  
 fracPass **196, 248**  
 GUBCovers **248**  
 gubCovers **202**  
 heurFreq **190, 248**  
 implBd **202, 248**  
 intSolLim **195, 248**  
 itLim **183, 248**  
 MIPEmphasis **178, 248**  
 MIPOrdInd **192, 248**  
 MIPOrdType **190, 249**  
 MIPThreads **197, 249**  
 MIRCut **204, 249**  
 netEpOpt **210, 249**  
 netEpRHS **210, 249**  
 netFind **210, 249**  
 netItLim **210, 249**  
 netPPriInd **209, 249**  
 nodeFileInd **197, 249**  
 nodeLim **194, 249**  
 nodeSel **191, 250**  
 objDif **199, 250**  
 objLLim **184, 250**  
 objULim **184, 250**

perInd **184, 250**  
 perLim **182, 250**  
 PiecewiseCuts **179, 250**  
 pPriInd **181, 250**  
 preCompress **186, 250**  
 preDual **188, 251**  
 preInd **187, 251**  
 prePass **187, 251**  
 preslNd **192, 251**  
 priceLim **182, 251**  
 probe **192, 251**  
 reduce **188, 251**  
 reInv **182, 251**  
 relaxPreInd **187, 251**  
 relObjDif **200, 251**  
 RINSHeur **193, 252**  
 scaInd **181, 252**  
 singLim **182, 252**  
 startAlg **191, 252**  
 strongCandLim **195, 252**  
 strongItLim **195, 252**  
 strongThreadLim **195, 252**  
 subAlg **191, 252**  
 subMIPNodeLim **197, 252**  
 symmetry **186, 252**  
 tiLim **176, 253**  
 treLim **196, 197, 253**  
 varSel **191, 253**  
 workDir **179, 253**  
 workMem **179, 253**  
 Paste button **31**  
 Paste command **27**  
 pathname  
     for compiled models **173**  
     for include files **172**  
     of current file **23**  
 perInd parameter **184, 250**  
 perLim parameter **182, 250**  
 PiecewiseCuts parameter **179, 250**  
 pPriInd parameter **181, 250**  
 preCompress parameter **186, 250**  
 preDual parameter **188, 251**  
 preInd parameter **187, 200, 251**  
 prePass parameter **187, 251**  
 preslNd parameter **192, 251**

priceLim parameter **182, 251**  
 probe parameter **192, 251**  
 proceeding through executions **71**  
 processing directives **40, 71**  
 production planning tutorial  
     alternative data file **73**  
     closing the project file **74**  
     continuing the execution **71**  
     creating the project **54, 59**  
     displaying results **70**  
     editing area with model file **56**  
     examining the solution **63**  
     executing the project **62**  
     Main window after executing **72**  
     Main window with data file **58**  
     Optimization notebook page **64**  
     problem description **52**  
     product.dat file **53**  
     product.mod file **52**  
     product.prj file **59**  
     Solutions notebook page **63**  
     Solver notebook page **65**  
 Project menu  
     Add New Data File **28**  
     Add New Model File **28**  
     Browse Model **28**  
     Close Project **28**  
     Insert Existing Data File **28**  
     Insert Existing Model File **28**  
     Project Options **28**  
     Save the Project **28**  
     Save the Project As **28**  
     Set as Active Project **28**  
 Project Options command **28**  
 Project Options dialog box **60**  
 project options, setting **60, 61**  
 project tree  
     description **23, 54**  
     displayed in workspace **74**  
     for alloc.prj **133**  
     for car.prj **83**  
 projects  
     adding existing data files **57**  
     adding existing model files **55**  
     adding new data files **58**

    adding new model files **58**  
     closing **74, 92**  
     creating **54, 59**  
     executing **62, 85, 137, 142**  
     opening **83**  
 Projects page **133**  
 projects, description **35**

## Q

Quit command **26**

## R

reading from a database **224**  
 Rebuild Browser Information button **32**  
 Recent Files command **26**  
 Recenter command **27**  
 Redo button **31**  
 Redo command **26**  
 reduce parameter **188, 251**  
 referenced files in OPL/OPLScript **172**  
 regional settings  
     database restriction in Japanese version **21, 213**  
     overriding default Japanese settings **21**  
 reInv parameter **182, 251**  
 relaxPreInd parameter **187, 251**  
 relobjDif parameter **200, 251**  
 removing a file from a project **75**  
 Replace command **27**  
 resources, definition **94**  
 results, displaying **90**  
 REW file format **177**  
 RINSHeur parameter **193**  
 RINSHeur parameter **252**  
 RLP file format **177**  
 Run button **32**  
 Run command **28**

## S

SAV file format **177**  
 Save All command **26**  
 Save All Files button **31**  
 Save and Close Project button **31**



- Save As command **26**
- Save command **26**
- Save Editor Content button **31**
- save preferences **47**
- Save the Project As command **28**
- Save the Project command **28**
- saving a file **31**
- saving project options **61**
- saving several files **31**
- scaInd parameter **181, 252**
- scheduling-specific dynamic display **113**
- scripts
  - debugging **236**
  - description **35, 232**
  - executing **234**
  - executing step by step **236**
  - opening a script file **233**
  - setting as active **76**
  - stepping out from a loop **237**
- search procedure
  - Best First Search **165**
  - Depth First Search **165**
  - Depth-bounded Discrepancy Search **165**
  - Interleaved Depth First Search **165**
  - Slice-Based Search **165**
- search strategy
  - dichotomic **165**
  - standard **165**
  - visualizing **147**
- search tree **148**
  - exploration order **150**
  - visualizing **148**
- Select All command **27**
- Set as Active Project command **28**
- setting
  - active model **76**
  - active script **76**
  - advanced options **171**
  - constraint programming options **165**
  - output options **169**
  - project options **60, 61**
- setting the active project **75**
- setup notebook page **91**
- singLim parameter **182, 252**
- Solutions notebook page **63, 100**

- solutions, examining **88, 100**
- Solver notebook page **65, 109**
- solving algorithms used **65**
- Sorted Activities notebook page **106**
- spreadsheet, paste results to **88**
- square example **120**
- Stack window **139, 144**
- stack window **124**
- stand-alone models and scripts **74**
- startAlg parameter **191, 252**
- status bar description **23**
- status box description **24**
- status colors **24**
- status messages
  - after execution **88**
  - during execution **62, 85, 98**
- Step Out command **29**
- stepping in model (debug mode) **142**
- stepping in script (debug mode) **236**
- stepping out of a script **32, 237**
- strongCandLim parameter **195, 252**
- strongItLim parameter **195, 252**
- strongThreadLim parameter **195, 252**
- subAlg parameter **191, 252**
- subMIPNodeLim parameter **197, 252**
- symmetry parameter **186, 252**

## T

- terminating
  - executions **152**
  - ILOG OPL Studio **40, 162**
- text-editor
  - customizing **46**
  - Emacs mode **43**
  - how to use **41**
  - Multi-Document Interface **41**
  - quick reference **43**
  - visual mode **43, 46**
- tiLim parameter **176, 253**
- tool bar description **23**
- tree
  - in project workspace **74, 133**
  - search layout **151**
  - visualizing search tree **148**

`treLim` parameter **196, 197, 253**

## U

Uncomment command **27**

Undo button **31**

Undo command **26**

updating a database

    creating a new table **225**

    viewing the result **226**

    viewing the result from a model **229**

## V

variables, examining results **86**

`varSel` parameter **191, 253**

Vellino example (OPLScript) **232**

View Choice Stack button **32**

View menu

    Choice Stack **28**

    Inspect Current Node **28**

    Output **28**

    Workspace **27**

viewing tables in a database **218**

## W

waiting state, blink option **171**

Window menu

    Cascade **29**

    Tile Horizontally **29**

    Tile Vertically **29**

windows

    docking **34**

    floating **33**

work space, description **23**

`workDir` parameter **179, 253**

`workMem` parameter **179**

`workMemr` parameter **253**

Workspace command **27**