



Changing the rules of business™

ILOG OPL Development Studio

From OR to OPL and ODM

Copyright

COPYRIGHT NOTICE

Copyright © 1987-2008, by ILOG S.A. and ILOG Inc. All rights reserved.

ILOG OPL Development Studio 6.0.1, ILOG CPLEX 11.1, ILOG CP Optimizer 2.0, ILOG Concert Technology 2.6, ILOG JViews 8.1, ILOG DBLink 5.2 Copyright © 1997 – 2008, by ILOG S.A., 9 Rue de Verdun, 94253 Gentilly Cedex, France, and ILOG, Inc., 1195 W. Fremont Ave., Sunnyvale, California 94087-3832, USA. All rights reserved.

General Use Restrictions

This document and the software described in this document are the property of ILOG and are protected as ILOG trade secrets. They are furnished under a license or nondisclosure agreement, and may be used or copied only within the terms of such license or nondisclosure agreement.

No part of this work may be reproduced or disseminated in any form or by any means, without the prior written permission of ILOG S.A, or ILOG Inc.

Trademarks

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.

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

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 United States and other countries.

This product includes software developed by the University of California, Berkeley and its contributors.

Open Source Software

This software and the accompanying materials include programs made available by the Eclipse Foundation under the terms of the Eclipse Public License v1.0 which accompanies this distribution. The Eclipse Public License version 1.0 is available at <http://www.eclipse.org/legal/epl-v10.html>.

Acknowledgement

The language manuals are based on, and include substantial material from, The OPL Optimization Programming Language by Pascal Van Hentenryck, © 1999 Massachusetts Institute of Technology.

Table of contents

From OR to OPL and ODM.....	7
A quick start to OPL.....	9
What is in the Quick Start?.....	11
The Main window.....	12
Understanding OPL projects.....	13
Projects / folders.....	14
Data files.....	15
Model files.....	16
Settings files.....	18
Run configurations.....	19
Modeling the P-Median problem with OPL.....	20
Two solving engines.....	23
Debugging and dealing with error messages.....	24
Displaying solutions.....	25
Summary: what you can do with the OPL IDE.....	26
From problem solving to what-if scenarios: ILOG ODM.....	27
ODM Overview.....	28
Rapid development with tightly integrated OPL and ODM.....	29
How OPL and ODM integrate.....	31
Index.....	33

From OR to OPL and ODM

Provides an overview of ILOG OPL and its main features. Make sure you read *How to use the documentation* for details of prerequisites, conventions, documentation formats, and other general information.

In this section

A quick start to OPL

A tour of OPL for users who are familiar with optimization problems but not with the OPL product.

From problem solving to what-if scenarios: ILOG ODM

Presents ILOG Optimization Decision Manager, the ILOG solution for the development and deployment of optimization-based planning and scheduling applications.

A quick start to OPL

A tour of OPL for users who are familiar with optimization problems but not with the OPL product.

In this section

What is in the Quick Start?

This section is for OR experts who are not familiar with ILOG OPL and who want a fast introduction to the product and its concepts.

The Main window

Presents a graphical view of the main window of the OPL IDE, and a brief description of its primary controls and elements.

Understanding OPL projects

ILOG OPL Development Studio handles project files, data files, model files, setting files, and run configurations.

Modeling the P-Median problem with OPL

Explains how to model the well-known P-Median problem in OPL.

Two solving engines

Presents the two solving engines that underlie OPL: CPLEX and CP Optimizer.

Debugging and dealing with error messages

Describes how OPL checks for and displays syntax errors and solving errors.

Displaying solutions

Provides a short description of how you can view solutions to your problems in OPL, both while the solver is running and after the solve has finished.

Summary: what you can do with the OPL IDE

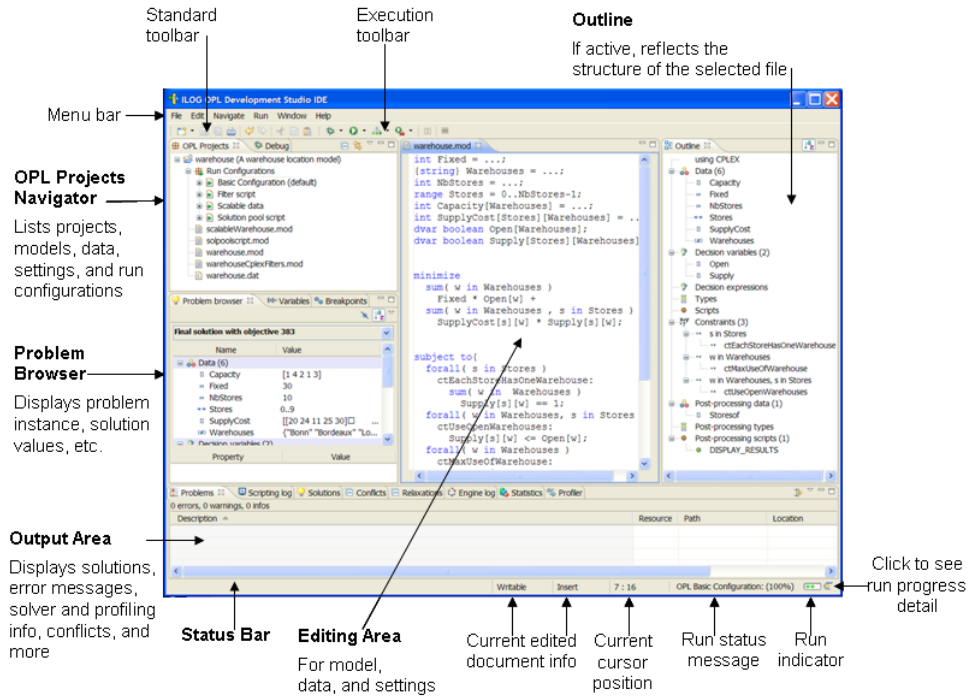
Summarizes the features and capabilities of OPL.

What is in the Quick Start?

This Quick Start is written for OR experts who are not familiar with the ILOG OPL approach for development and deployment of optimization models. It starts from a well known, hands-on example based on the *P-Median* problem. More detailed information on the concepts, terms, and procedures presented in this section is provided throughout the OPL documentation set, in particular in the *Glossary* and in *Getting Started with the IDE*.

The Main window

Tooltips appear when you move the pointer over most elements of the main window.



Understanding OPL projects

ILOG OPL Development Studio handles project files, data files, model files, setting files, and run configurations.

In this section

Projects / folders

Defines what a project is in OPL.

Data files

Describes how data (.dat) files are used in OPL.

Model files

Describes how model (.mod) files are used in OPL.

Settings files

Describes how settings (.ops) files can be used to change the default values of OPL.

Run configurations

Describes how different run configurations can be defined in OPL to support multiple execution configurations.

Projects / folders

ILOG OPL uses the concept of a *project* to associate a model (.mod) file with, usually, one or more data (.dat) files and one or more settings (.ops) files.

A project containing only a single model file is valid; data and settings files are optional. However, one project can contain several sets of model, data and settings files, the relationships between them maintained via *run configurations*.

In previous releases of OPL, these OPL projects were defined (and the relationships between the files were maintained) in a special project (.prj) file. As of OPL 6.x, OPL projects are defined by the *folders* in which they are stored. One project folder can contain a single model, or it can contain multiple model and data files handled by run configurations.

The *model file* declares data elements but does not necessarily initialize them. The *data files* contain the initialization of data elements declared in the model.

The .project file in the root folder for the OPL project organizes all the related model, data and settings files. Run configurations, which are maintained in an .oplproject file, also provide a convenient way to maintain the relationship between related files and runtime options for the environment (see also the *Run configurations* section).

When you are about to write a new model in ILOG OPL Development Studio, choose **File>New>OPL Project**. The dialog box that appears allows you to name your project, give your project a description, and choose whether you want to create a data file or a settings file. The description of the project may be useful later to better differentiate projects with similar names.

In OPL 6.x, each project name has to be unique. Duplicate project names are not allowed, wherever the project folders are stored on the file system.

A minimal project has:

- ◆ one model file
- ◆ one default run configuration referencing that same model file

A typical project has:

- ◆ one or more model files
- ◆ any number of data files or no data file
- ◆ any number of settings files or no settings file
- ◆ one or more run configurations referencing various combinations of those model, data, and settings files. (A run configuration cannot have more than one model file.)

Data files

You can organize large problems better by separating the model of the problem from the instance data, each set of data stored in a separate data file.

In this case, you store the instance data in one or more data files (.dat). Data files (.dat) store the actual values of the data used in the model. If you declare the data as suggested in this tutorial, your data file will look something like this:

```
n = 3;  
  
c = [[0.0 1.5 2.3]  
     [1.5 0.0 3.7]  
     [2.3 3.7 0.0]];
```

Each data file may specify one or more connections to data sources, such as a relational database or a spreadsheet, to read and write data. From the IDE, you can export external data and internal data to a .dat file, which you can later use as input. Only the data actually used in the model is exported to data files.

Model files

Model (.mod) files contain all your OPL statements. The data and the objective function are not mandatory and there may be more optional components, such as scripting statements. Note that you can also generate a model file in a compiled form (.opl) from the IDE for execution through the OPL interface libraries (see *Generating a compiled model*). The components of a model file are covered in the following sections.

Declarations of data

Data declarations allow you to name your data so that you can reference them easily in your model. For example, if your data in a table define the cost of shipping one unit of material from location i to location j , you might want to call your item of data $cost_{ij}$ where $i=1, \dots, n$ and $j=1, \dots, n$ and n is the number of locations in your model. You tell OPL that your model uses this data by declaring:

```
int n = . . . ;  
float cost[1..n][1..n] = . . . ;
```

The . . . (ellipsis) means that the actual values for your table are located in a data file, which must be listed in the current project.

You could also list the data explicitly in the model file. However, it is recommended that you construct model files without specifying values for data so that you can later easily solve many instances of the same model by simply changing the data file. See also the Run configurations section.

Note that the `int` type declared means that the numbers in the data file must be integers. If the numbers in the data file are floating-point numbers, use the `float` type instead.

Declarations of decision variables

In OPL context, as opposed to ILOG Script and to the general programming context, variables are decision variables. Declarations of decision variables name and give the type of each variable in the model. For example, if you want to create a variable that equals the amount of material shipped from location i to location j , you can create a variable named $ship_{ij}$.

```
dvar float+ ship[1..n][1..n];
```

That statement declares an array of non-negative floating-point variables. (That is what `float+` means). The `dvar` keyword indicates that you are declaring a decision variable.

An objective function

The objective function is a function that you want to optimize. This function must consist of variables and data that you have declared earlier in the model file. The objective function is introduced by either the `minimize` or the `maximize` keyword. For example,

```
minimize sum(i,j in 1..n) cost[i][j]*ship[i][j];
```

That statement indicates that you want to minimize the sum of the shipping costs for each origin-destination pair.

Constraints

Constraints indicate the conditions necessary for a feasible solution to your model. You declare constraints within a `subject to` block. For example,

```
subject to {  
    forall(j in 1..n) sum(i in 1..n) ship[i][j] == demand[j];  
}
```

That statement declares one set of constraints. There is a constraint for each destination. (That is what the `forall` keyword indicates.) The constraint for each destination states that the sum of material shipped to that destination must equal the demand at that destination. The symbol `==` indicates equals within a constraint block. The symbol `<=` indicates less than or equal to. The symbol `>=` indicates greater than or equal to.

Settings files

Settings files (.ops) are where your user-defined values are stored when you decide to change the default values of OPL language options, constraint-programming (CP Optimizer) parameters, or mathematical-programming (CPLEX) parameters.

OPL settings apply only to the model included in the run configuration, not to the submodels loaded and solved.

Run configurations

Run configurations are not file types, properly speaking, but rather a way of handling model, data, and settings files within a project.

Basically, a run configuration is a variation of a given project for execution purposes. It combines at least a model file and, optionally, one or more data files and one or more settings files within the project, while addressing the same mathematical problem. You can define as many run configurations as you need within a given project. Typically, you use run configurations to test, improve, and fine-tune your OPL projects.

For example, you can:

- ◆ keep two sets of data: a simple one for quick prototyping and a larger one to work closer to your business case;
- ◆ keep one configuration for each set of MP options (CPLEX parameters) that makes sense for your problem.

Practically, run configurations appear as sublevels in the Projects tree.

Modeling the P-Median problem with OPL

The *P-Median* problem is a well known problem in Operations Research. The problem can be stated very simply, like this: given a set of customers with known amounts of demand, a set of candidate locations for warehouses, and the distance between each pair of customer-warehouse, choose P warehouses to open that minimize the demand-weighted distance of serving all customers from those P warehouses.

A standard textbook would probably write out the P-Median problem like this:

Data

- P = number of facilities to locate
 I = set of customers
 J = set of candidate locations for warehouses
 h_i = demand for customer i
 d_{ij} = distance between customer i and candidate warehouse j

Decision variables

- x_j = 1 if candidate warehouse j is used; 0 otherwise
 y_{ij} = 1 if demand for customer i is satisfied by warehouse j ; 0 otherwise

Objective

Minimize the demand-weighted distance of delivering to customers.

Or, expressed as an equation:

$$\min \sum_{i \in I} \sum_{j \in J} h_i d_{ij} y_{ij}$$

Constraints

- 1st constraint makes sure that each customer is served by exactly one warehouse.
2nd constraint makes sure that P warehouses are built.

3rd constraint makes sure that a customer is not served by an unopened warehouse.

Or, expressed as equations:

$$\sum_{j \in J} y_{ij} = 1$$

$$\sum_{j \in J} x_j = P$$

$$y_{ij} \leq x_j \quad i \in I \quad j \in J$$

$$x_j \in \{0, 1\} \quad j \in J$$

$$y_{ij} \in \{0, 1\} \quad i \in I \quad j \in J$$

The construction of an OPL model file (.mod) follows the same format exactly. The P-Median problem can be formulated in ILOG OPL Development Studio like this:

```
//Data
int P = ...;
{string} Customers = ...;
{string} Warehouses = ...;
int Demand[Customers] = ...;
float Distance[Customers][Warehouses] = ...;

//Variables
dvar boolean OpenWarehouse[Warehouses];
dvar boolean ShipToCustomer[Customers][Warehouses];

//Objective
minimize
    sum( c in Customers , w in Warehouses )
        Demand[c]*Distance[c][w]*ShipToCustomer[c][w];

//Constraints
subject to {
    forall( c in Customers )
        ctShip:
            sum( w in Warehouses )
                ShipToCustomer[c][w] == 1;

    ctOpen:
        sum( w in Warehouses )
            OpenWarehouse[w] == P;

    forall( c in Customers , w in Warehouses )
        ctShipOpen:
```


```
ShipToCustomer[c][w] <= OpenWarehouse[w];  
}
```

Notice that:

- ◆ the names of the customers and warehouses are specified as sets of strings in the data file. (That is what the notation {string} means.)
- ◆ it is also possible to use sets of integers as the labels for the customers and warehouses. You can also use ranges instead of sets. The demand and distance data remain the same, regardless of the format of the labels for customers and warehouses.
- ◆ the . . . (ellipsis) syntax means that the data is initialized externally, that is, from a data file (pmedian.dat).

```
P=2;  
Customers={"Albert","Bob","Chris","Daniel"};  
Warehouses={"Santa Clara","San Jose","Berkeley"};  
Demand=[100,80,80,70];  
Distance=  
[[ 2 , 10 , 50 ],  
 [ 2, 10, 52 ],  
 [ 50, 60 , 3],  
 [ 40 , 60 , 1]];
```

Two solving engines

After the model, data, and settings files are complete, use the **Run** button  or the context menus in the OPL Projects Navigator to run your model.

The time it will take to solve a model naturally depends on the size and complexity of the model. Typical textbook problems like the P-Median problem usually solve very quickly. If the solver finds a feasible solution, it displays the solution in the **Solutions** tab of the Output Area. (See *The Main window*.) The solver will continue working until an optimal solution is found or until you click the **Abort** button.

ILOG OPL Development Studio 5.2 and later supports two solving engines:

- ◆ the CPLEX engine for mathematical programming is used by default when you run your project if your model does not start with the statement `using CP;`. This is the case of the P-Median problem described in the *Modeling the P-Median problem with OPL*.
- ◆ the CP Optimizer engine for constraint programming is called if your model starts with the statement `using CP;`. You can write, edit and solve the model from the OPL IDE, from the `oplrun` command line tool, and from the APIs.

You will find more information on CP modeling and solving in the documentation set. In particular, *Constraint programming versus mathematical programming* in the *Language User's Manual* presents the differences between MP models and CP models.

Debugging and dealing with error messages

ILOG OPL Development Studio checks for errors in the model and data files.

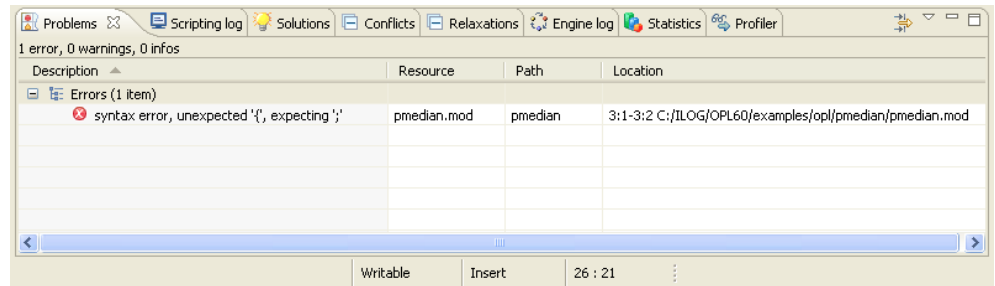
Syntax errors

Syntax and semantic errors are displayed dynamically in the Issues tab as you type.

For example, a common error is to forget to put a semicolon at the end of a statement. If you omit the semicolon at the end of the line

```
int P = ...;
```

the **Problems** tab displays the description, location, and source file of the error.



Generally, error messages will look similar to this example.

Solving errors

Immediately after you run your project, OPL checks for errors that prevent the solver from running. If such errors are found, one or more error messages will be displayed in the **Problems** tab. (See *The Main window* section.)

Displaying solutions

It is possible for you to view solutions while the solver is running as well as after it has finished. In addition to the **Solutions** tab of the Output Area, you can view a solution in tabular form through the **Problem Browser**. (See *The Main window*.) If your model expresses a MIP problem that generates feasible solutions, you can see the solution pool in the Problem Browser and further populate it with more nonoptimal solutions.

You can see variable values in the Problem Browser, which also contains information about data structures, data values, labeled constraints and sensitivity data, as well as postprocessing data.

Summary: what you can do with the OPL IDE

ILOG OPL IDE is an integrated development environment (IDE) for mathematical programming, constraint programming, and combinatorial optimization applications in general. It is the graphical user interface (GUI) for the Optimization Programming Language and ILOG Script for OPL, its scripting language.

With the ILOG OPL IDE you can:

- ◆ Create and modify project files, as well as model and data, using the editing capabilities
- ◆ Create and modify settings files to apply language options, mathematical programming parameters and constraint programming parameters
- ◆ Execute a project
- ◆ Visualize OPL results in text or tabular form
- ◆ Search for relaxations of variables and constraints, and for conflicts between constraints in infeasible MP models
- ◆ Identify the time and memory consumed for the execution of a project
- ◆ Work with ILOG Script for OPL, the scripting language
- ◆ Debug scripts using the debug facilities
- ◆ Generate a compiled model
- ◆ Generate external data files or internal data files to various formats
- ◆ Visualize the state of variables at some point during the search for a solution
- ◆ Connect to a database or to a spreadsheet to read and write data
- ◆ Set preferences on the aspect of the IDE
- ◆ and carry out many other tasks

From problem solving to what-if scenarios: ILOG ODM

Presents ILOG Optimization Decision Manager, the ILOG solution for the development and deployment of optimization-based planning and scheduling applications.

In this section

ODM Overview

Presents a short overview of ILOG Optimization Decision Manager (ODM).

Rapid development with tightly integrated OPL and ODM

Describes the features and benefits of ILOG ODM.

How OPL and ODM integrate

Describes the overall ILOG optimization suite and interactivity of its components — OPL, ODM, CPLEX, and CP Optimizer.

ODM Overview

ILOG OPL Development Studio is integrated with a companion product called ILOG Optimization Decision Manager (ODM). ILOG ODM is both a tool for application development and a runtime environment, and the combined product is a complete solution for the development and deployment of optimization-based planning and scheduling applications.

Applications built with ILOG ODM allow users to adjust assumptions, operating constraints, and goals for planning and scheduling resources, and see the results in familiar business terminology, providing extensive what-if analysis and scenario comparison features “out of the box.”

In essence, you supply the mathematical model in OPL, and ODM supplies the GUI and the interactivity with your model that makes it easy to use for operations managers and planners. For more information on the features of an ODM application, and to see a demo, please go to <http://odm.ilog.com>.

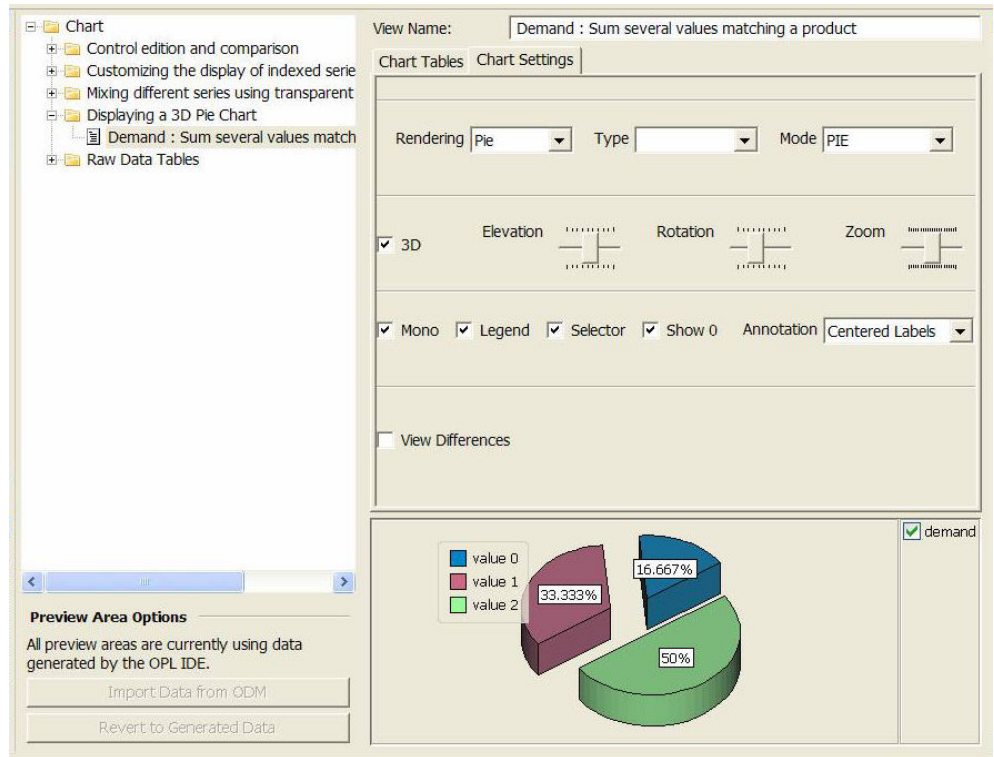
Rapid development with tightly integrated OPL and ODM

Developing an application with ILOG OPL and ILOG ODM is very easy and productive.

The integration comes with a wizard that automatically generates an ODM application based on the structure of your OPL model.

- ◆ OPL data structures such as arrays and tuple sets are mapped to input data tables and graphical views in ODM. Because they are editable, end users can then play out scenarios in ODM based on data changes.
- ◆ Mathematical constraints from your OPL model can be exposed as ODM *requirements* (soft constraints) with priorities, allowing a business user to make trade-offs between conflicting business requirements and alternative relaxations.
- ◆ The objective(s) of your mathematical model are mapped to ODM *goals*, which allow interactive goal programming and balancing between multiple objective criteria.
- ◆ Decision variables in the model are represented as solution views in ODM, and can be configured to show aggregate values and key performance indicators (KPIs), as well as providing exporting to Excel and integration with reporting tools.

The generated ODM application is configurable through editors inside OPL, and as the model evolves it can be extended to take new business requirements into account. The following figure shows how a chart view in ODM is configured in OPL.



Configuring an ODM chart view in OPL

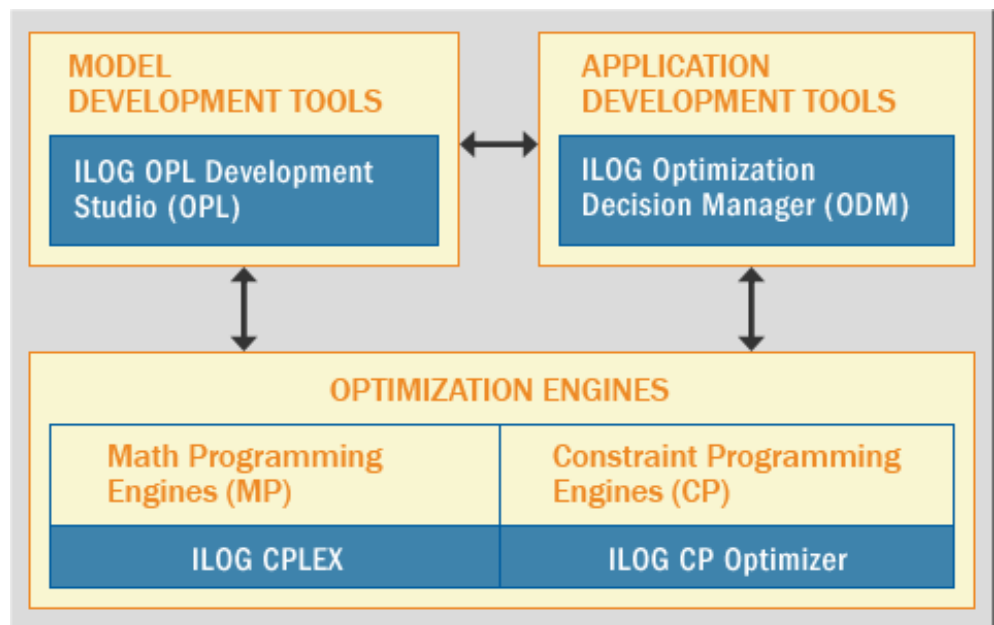
Developing an application with ILOG OPL and ILOG ODM also allows business users to take part in the rapid prototyping and iterative development needed to strengthen definitions of requirements, refine models, perfect scenario parameters, and examine KPIs, solution views, and business goals.

Note: If you are reading this because you are a current OPL user but without access to ODM, please contact your ILOG sales representative.

How OPL and ODM integrate

ILOG provides a complete set of tools for building custom optimization-based decision support applications, which includes.

- ◆ **ILOG CPLEX** – A mathematical programming engine that solves the full range of LP and MIP problems encountered in the real world.
- ◆ **ILOG CP Optimizer** – A constraint programming engine that solves complex scheduling and routing problems.
- ◆ **ILOG OPL Development Studio** – A complete IDE for rapid development and deployment of optimization models.
- ◆ **ILOG Optimization Decision Manager (ILOG ODM)** – A tool for developing interactive decision support applications with out-of-the-box “what-if” analysis, scenario comparisons and goal programming.



OPL, ODM, CPLEX and CP Optimizer

ILOG makes optimization more widely accessible than ever before, because it offers such a wide range of capabilities across several fundamental dimensions, typically:

- ◆ **Supporting many types of optimization applications:** From off-line strategic planning applications through detailed scheduling to real-time operational applications.

- ◆ **Supporting the complete model development and application life-cycle:** Complete support for the application development process, from quick prototyping through model and application refinement, to application deployment and maintenance.
- ◆ **Supporting multiple personas:** OR and IT professionals, together with operations and business managers, can now collaborate in entirely new ways to rapidly generate custom decision-support applications.

Index

- A**
- acknowledgements **2**
- C**
- copyright information **2**
 - general use restrictions **2**
 - open source software **2**
 - trademarks **2**
- D**
- data file **14**
- F**
- folder **14**
- M**
- model file **14**
- O**
- ODM
 - overview **28**
- P**
- project **14**
- R**
- run configurations **14**
- S**
- settings file **14**