Completely new to CPLEX Optimization Studio?

A first contact with OPL, the Optimization Programming Language, and the integrated development environment (IDE).

→ **Who this document is for**
Intended for optimization experts who are using IBM® ILOG® CPLEX Optimization Studio for the first time.

→ **Brief overview of CPLEX Optimization Studio**
CPLEX Optimization Studio consists of a modeling language, two optimization engines to solve the models, and an integrated development environment to test and debug the models.

→ **Analyzing a simple OPL model**
Analyzes a simple OPL model and describes how to execute the model in the IDE.

→ **First steps with the IDE**
Briefly describes how to begin working with an OPL model in the IDE, after the IDE has been installed and launched.

→ **Running a model in the IDE**
Hands-on practice with a distributed example.

Parent topic:
→ Orientation Guide

Who this document is for

Intended for optimization experts who are using IBM® ILOG® CPLEX® Optimization Studio for the first time.

This topic is for optimization experts who know how to formulate mathematical models that represent business problems, and who are using CPLEX Optimization Studio for the first time. It introduces a simple model and describes how to work with projects and models in the integrated development environment (IDE). It ends with a hands-on exercise in the IDE.

Parent topic:
→ Completely new to CPLEX Optimization Studio?

Brief overview of CPLEX Optimization Studio

CPLEX® Optimization Studio consists of a modeling language, two optimization engines to solve the models, and an integrated development environment to test and debug the models.

CPLEX Optimization Studio is composed of:

- OPL, the Optimization Programming Language, used to write mathematical models.
- An integrated development environment (IDE) that enables you to develop and test the models.
- The CPLEX Optimizer engine, to find solutions to models that require mathematical programming techniques.
- The CP Optimizer engine, to find solutions to models that require constraint programming techniques.

Parent topic:

→ Completely new to CPLEX Optimization Studio?

Analyzing a simple OPL model

Analyzes a simple OPL model and describes how to execute the model in the IDE.

The business problem

A manufacturer has some products he wants to sell, and these products are manufactured with resources. The products can be made in either of two ways. They can be made inside the factory, where production consumes scarce resources and there is a cost per unit to manufacture the products, or they can be ordered from outside the factory. In the latter case there is no resource usage but there is a higher cost per unit to purchase the products. There is a constraint that all customer demand for the products must be satisfied, and the business goal is to minimize cost.

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.

The mathematical representation

Conventions used in this mathematical description

- Uppercase is used for data and constants
- Lowercase is used for variables

Data

- P is the number of products; p is an index ranging from 1 to P.
- R is the number of resources; r is an index ranging from 1 to R.
- $C_{p,r}$ is the amount of resource r consumed by product p when produced inside.
- D_p is the amount of demand for product p.
- $INSIDECOST_p$ is the cost of producing one unit of product p.
- $OUTSIDECOST_p$ is the cost of providing one unit of product p from an outside source.
- $CAPA_r$ is the maximal capacity of resource r.

Decision variables

- in_p is the amount of product p manufactured inside the plant (positive or null).
- out_p is the amount of product p provided by outside sources (positive or null).
-

The mathematical problem

$$\textbf{minimize} \left(\sum_{p=1}^{p=P} INSIDECOST_p in_p + OUSIDECOST_p out_p \right) \quad (1)$$

subject to

$$\forall r \ in\ 1..R, \quad \left(\sum_{p=1}^{p=P} C_{p,r} * in_p \right) \leq CAPA_r \quad (2)$$

$$\forall p \ in\ 1..P, \quad in_p + out_p \geq D_p \quad (3)$$

$$\forall p \ in\ 1..P, \quad in_p \geq 0, out_p \geq \mathbf{0} \quad (4)$$

The objective, line (1), is to minimize the total cost, computed as the sum over all products of both the internal cost incurred by actually producing the product and the cost of purchasing the product from an outside source.

The capacity constraint, line (2), states that the total required capacity cannot exceed the maximum for any resource.

The demand constraint, line (3), states that, for all products, the sum of products produced inside, plus those provided from outside sources, must be greater than the demand.

Line (4) states that we are only interested in solutions for which production amounts are positive.

The OPL code

OPL allows you to write a mathematical representation of your business problem that is separate from your data. An example of an OPL model, `production.mod`, is shown here. The terms in **bold** are OPL keywords.

```
{string} Products = ...;
{string} Resources = ...;

float Consumption[Products][Resources] = ...;
float Capacity[Resources] = ...;
float Demand[Products] = ...;
float InsideCost[Products] = ...;
float OutsideCost[Products]  = ...;

dvar float+ Inside[Products];
dvar float+ Outside[Products];

minimize
  sum( p in Products )
    ( InsideCost[p] * Inside[p] + OutsideCost[p] * Outside[p] );

subject to {
  forall( r in Resources )
```

```
    ctCapacity:
      sum( p in Products )
        Consumption[p][r] * Inside[p] <= Capacity[r];

  forall(p in Products)
    ctDemand:
      Inside[p] + Outside[p] >= Demand[p];
}
```

1. The first step in specifying an OPL model is to declare the data, and in this particular instance the first things you declare are the index sets, which represent the set of products and the set of resources.

   ```
   {string} Products = ...;
   {string} Resources = ...;
   ```

   These are defined as sets of strings. That is, the strings are the names of the products and the resources. On the right-hand side of the equal sign, the three dots are the OPL notation to indicate that you read this data from an external data source, which could be a data file, a spreadsheet, or a database. You do not have to specify the data within the model; the data is read in from a data file.

2. Next, you specify the recipe. How many units of each resource are required for each product?

   ```
   float Consumption[Products][Resources] = ...;
   ```

   This is an array, called “Consumption” with two indexes. One index is the set of products and the other is the set of resources. The notation “float” indicates a real number. So each entry in this array represents the amount of resource that is required to produce one unit of each product.

3. There is a capacity available for each resource, or an amount of each resource available.

   ```
   float Capacity[Resources] = ...;
   ```

   This is, again, a real number array, “Capacity”, that is indexed over the resources.

4. A quantity of each product is required. This is an array indexed by the products, called "Demand," and is also a real number.

   ```
   float Demand[Products] = ...;
   ```

5. And then there are the costs per unit for production, inside the factory and outside, of each product.

   ```
   float InsideCost[Products] = ...;
   float OutsideCost[Products]  = ...;
   ```

The product in this model could be anything. For example, it could be jewelry. So we could have a data file that goes with this particular model when we are producing jewelry that looks like this:

```
Products =  { "rings", "earrings" };  //index sets for
Resources = { "gold", "diamonds" };   //products and resources

Consumption = [ [3.0, 1.0], [2.0, 2.0] ];
Capacity = [ 130, 180 ];
Demand = [ 100, 150 ];
```

```
InsideCost = [ 250, 200 ];
OutsideCost  = [ 260, 270 ];
```

In that case, there are two products, rings and earrings, and two resources, gold and diamonds. The recipe calls for three units of gold and one diamond to produce each ring, or two units of gold and two diamonds for each earring. For the amount of resources available, there are 130 units of gold and 180 diamonds. The demand is for 100 rings and 150 pairs of earrings. The inside cost to produce the rings is $250 and to produce the earrings it is $200. The outside cost is $260 for rings and $270 for earrings.

However, in our distributed example, the products are pasta, and the `production.dat` file looks like this:

```
Products =  {"kluski", "capellini", "fettuccine"};
Resources = {"flour", "eggs"};

Consumption = [ [0.5, 0.2], [0.4, 0.4], [0.3, 0.6] ];
Capacity = [ 20, 40 ];
Demand = [ 100, 200, 300 ];
InsideCost = [ 0.6, 0.8, 0.3 ];
OutsideCost  = [ 0.8, 0.9, 0.4 ];
```

In this case there are three products - kluski, capellini and fettuccine - and there are two resources, flour and eggs. The consumption here is that producing a unit of kluski requires half a unit of flour and 0.2 units of eggs, and so forth for each of the other products. There is a capacity of 20 units of flour and 40 units of eggs. There is a demand of 100, 200, 300 for each of the three products -- kluski, capellini and fettuccine. And there is an inside and an outside cost to produce each product.

An important point to note here is that the model is the same for pasta, for jewelry, or any other product. The model does not depend on the data.

To return to the model, `production.mod`, we see that two decision variables are specified.

```
dvar float+ Inside[Products];
dvar float+ Outside[Products];
```

The two decision variables (`dvar`) are the inside production of each product and the outside production of each product, and they are specified as nonnegative real numbers, so we have a simple linear program here.

The actual specification of the model consists of the objective function and the constraints. The objective function in our example is to minimize the total cost of meeting the demand.

```
minimize
  sum( p in Products )
    ( InsideCost[p] * Inside[p] + OutsideCost[p] * Outside[p] );
```

This is the sum over each product in the product set of the inside cost of producing that product, times the inside production, plus the outside cost of producing that product, or acquiring that product, times the outside production.

The objective function is subject to two constraints.

```
subject to {
  forall( r in Resources )
    ctCapacity:
      sum( p in Products )
        Consumption[p][r] * Inside[p] <= Capacity[r];

  forall(p in Products)
    ctDemand:
      Inside[p] + Outside[p] >= Demand[p];
```

The first constraint (`ctCapacity`) is a capacity constraint on the resources. It says that, for each resource in the set of resources, the sum over all the products of the consumption of the resource for that product, times the inside production, has to be less than or equal to the capacity available for that resource.

The second constraint (`ctDemand`) is the demand constraint. It says that, for each product in the set of products, the inside production plus the outside production has to be at least as great as the demand. So, again, this is a simple linear programming problem.

Parent topic:
→ Completely new to CPLEX Optimization Studio?

First steps with the IDE

Briefly describes how to begin working with an OPL model in the IDE, after the IDE has been installed and launched.

1. To work with a model in the IDE, you must first create a project. The model and associated data are added to the project.
2. You create a model with the OPL modeling language in the editing area of the IDE. You then save the OPL code in a file with the extension `.mod`.
3. Usually, you create a data file in the IDE, and save the data in a file with the extension `.dat`. You can also have a simple model with the data inside the model.
4. Before executing a model, you need to create a run configuration, and add a model and data file to the run configuration. You then use the run command on a run configuration. If you want to test a model with different data files, you need to create a run configuration for each combination of model + data.
5. You can also add an optional settings file to a project, or to a run configuration. If you change the default values of settings, the new, user-defined, values are stored in the settings file with the extension `.ops`.

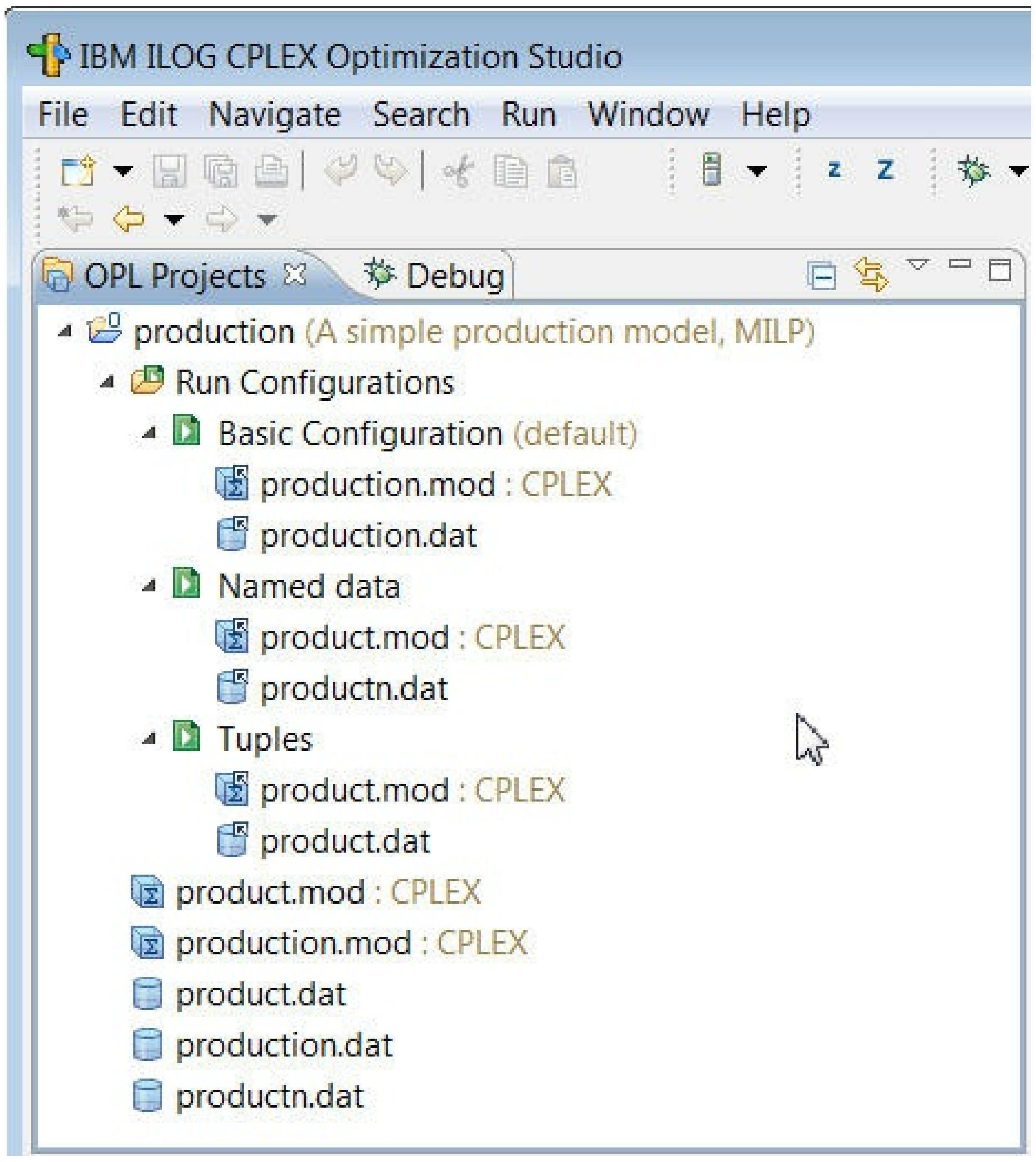
Summary

A project file and a run configuration are mandatory. A model is contained in a run configuration. It is the run configuration that is executed. A project can contain several run configurations. Data files and settings files are optional.

Note that many examples of projects with models and data files can be found in your installation directory, for example:

```
C:\Program Files\IBM\ILOG\CPLEX_Studio<version>\opl\examples\opl
```

In this document, we will use a distributed example of an OPL project called 'production'. Here you can see the project structure in the IDE.


IBM ILOG CPLEX Optimization Studio
File Edit Navigate Search Run Window Help
OPL Projects
Debug

production (A simple production model, MILP)
Run Configurations
Basic Configuration (default)
production.mod : CPLEX
production.dat
Named data
product.mod : CPLEX
productn.dat
Tuples
product.mod : CPLEX
product.dat
product.mod : CPLEX
production.mod : CPLEX
product.dat
production.dat
productn.dat

The 'production' project contains two model files and three data files. These files are used in three different run configurations: **Basic Configuration**, **Named data**, and **Tuples**. The comment next to the project name indicates that the problem to be solved uses Mixed-Integer Linear Programming. The models will call the CPLEX® solving engine.

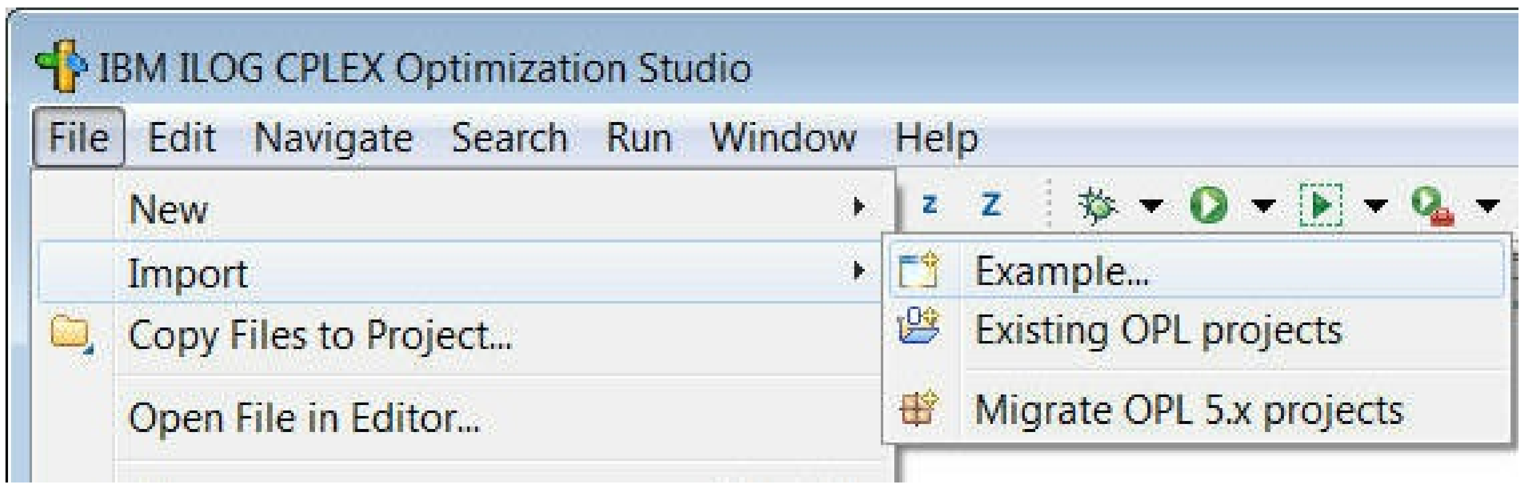
Parent topic:
→ Completely new to CPLEX Optimization Studio?

Running a model in the IDE

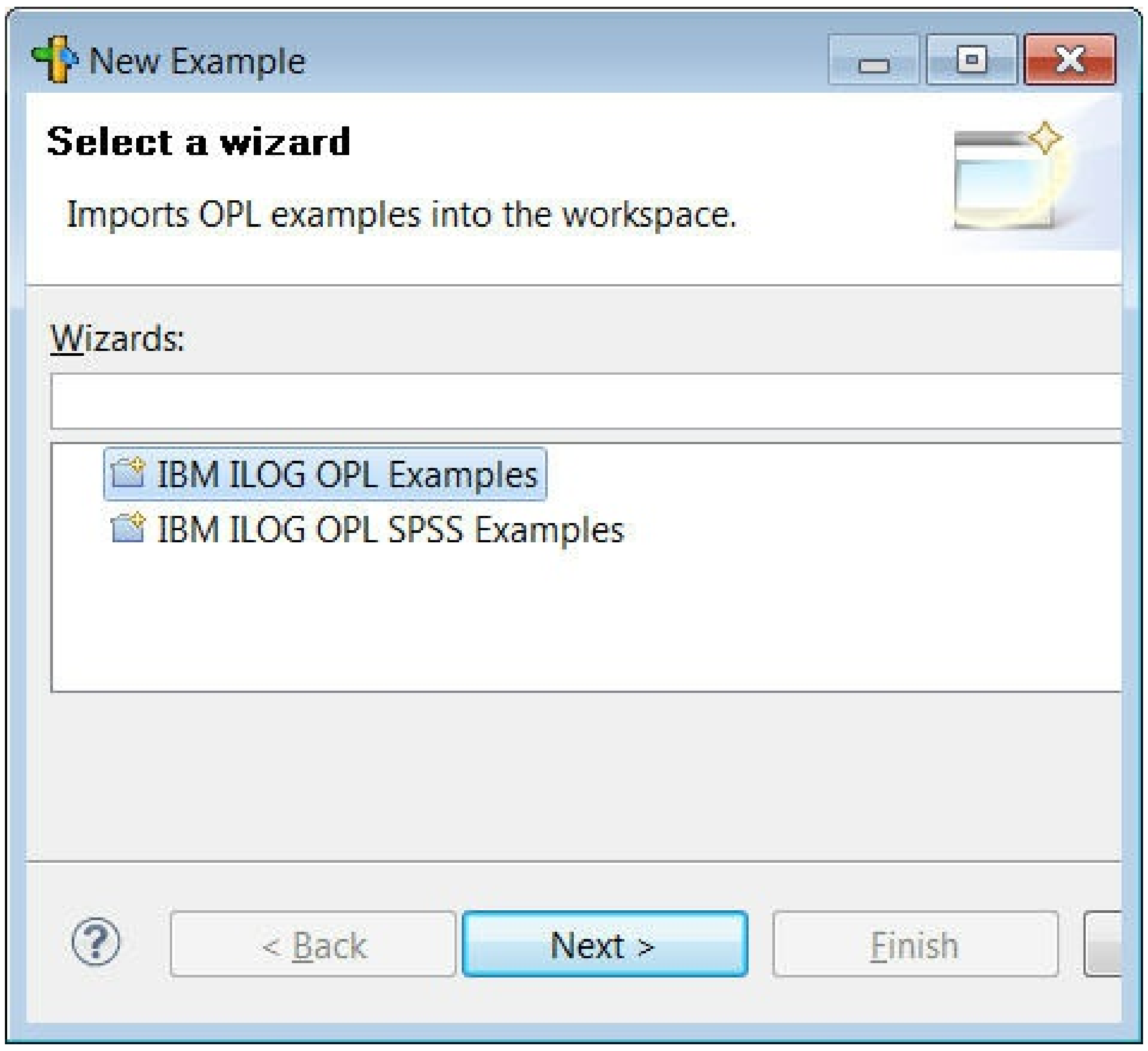
Hands-on practice with a distributed example.

1. Start the IDE, if you have not already done so. For example, on Windows platforms:
   Start > All Programs > IBM ILOG > CPLEX Optimization Studio`<version>` **> CPLEX Studio IDE**

2. Open the example wizard: **File > Import > Example**.

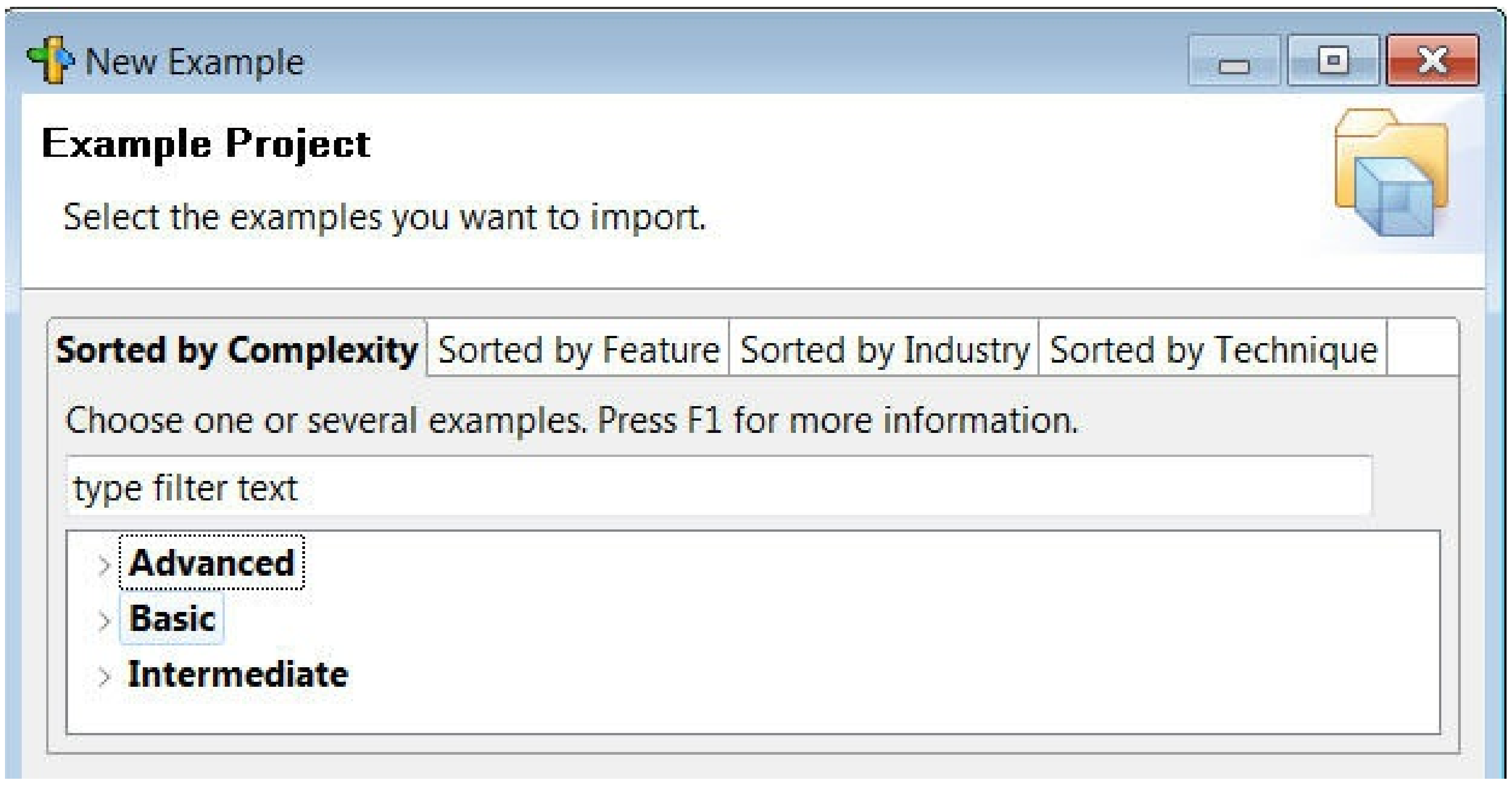


3. In the New Example wizard, select **IBM ILOG OPL Examples**, and click **Next**.



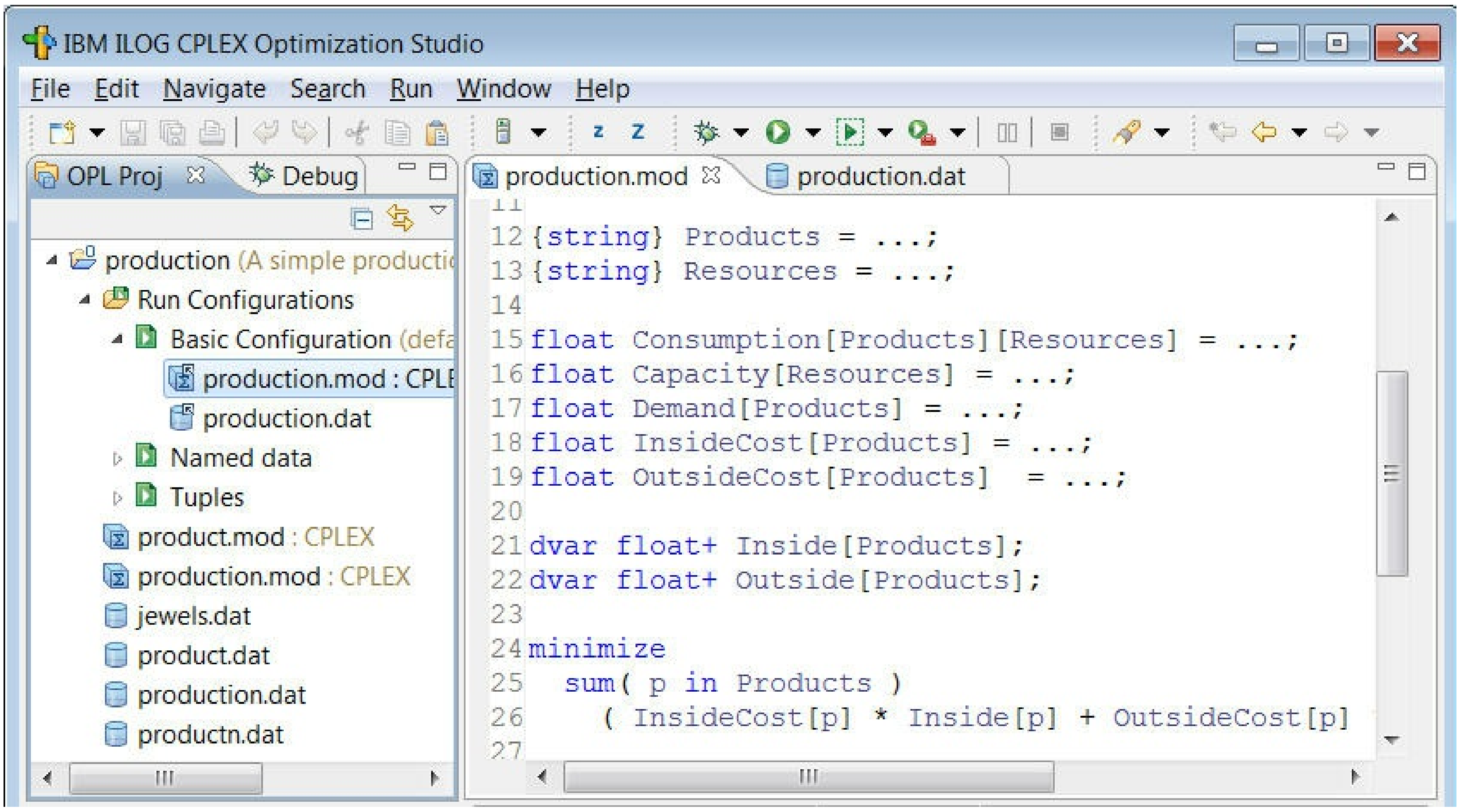
   The example wizard organizes the list of examples according to complexity (Advanced, Basic, Intermediary) or math programming technique.

4. Expand the **Basic** level, scroll down and select 'production', then click **Finish**.



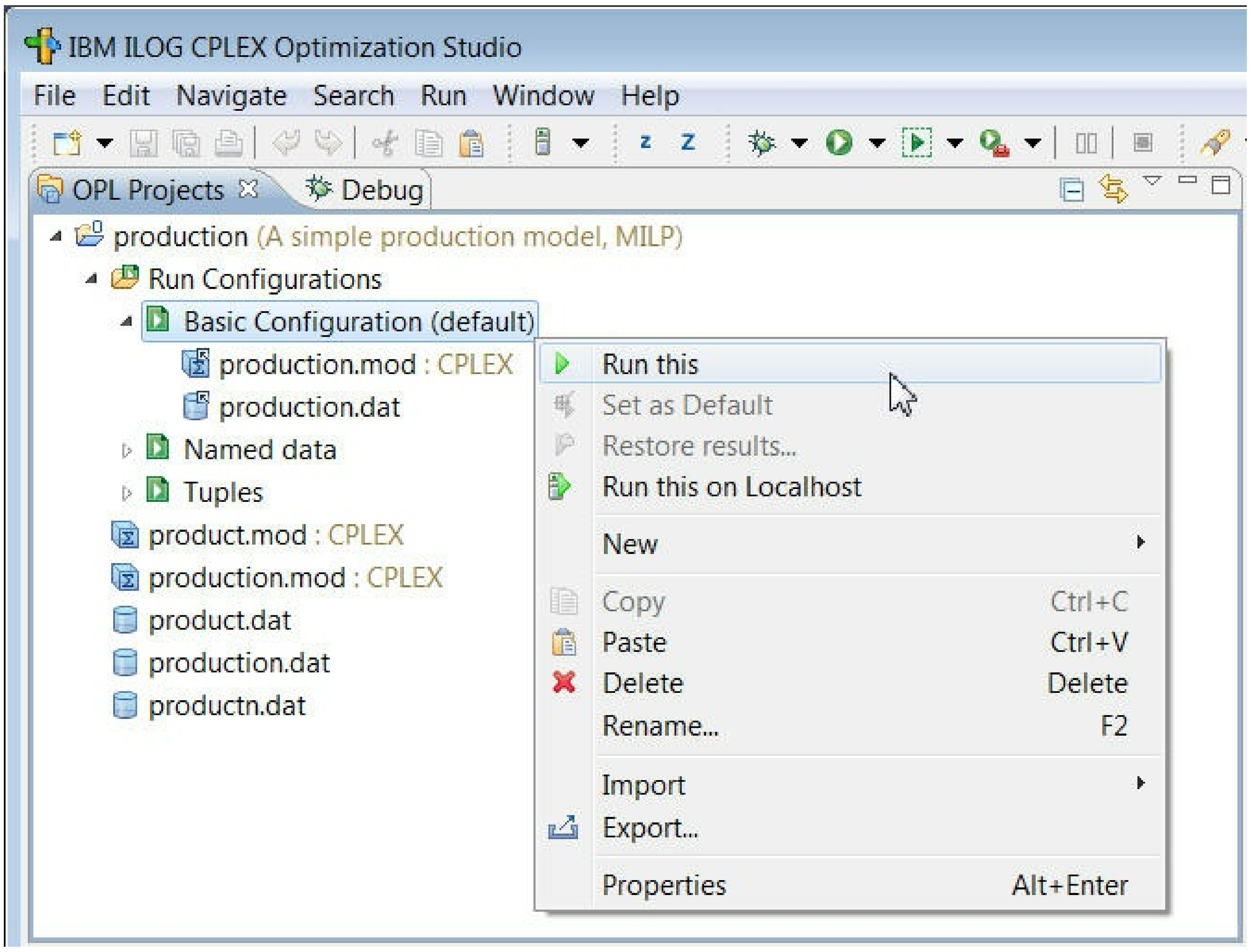
The collapsed 'production' project appears in the OPL Projects pane on the left.

5. Expand the 'production' project and the Run Configurations. In the Basic Configuration, double click the model and data files to display them in the editor.

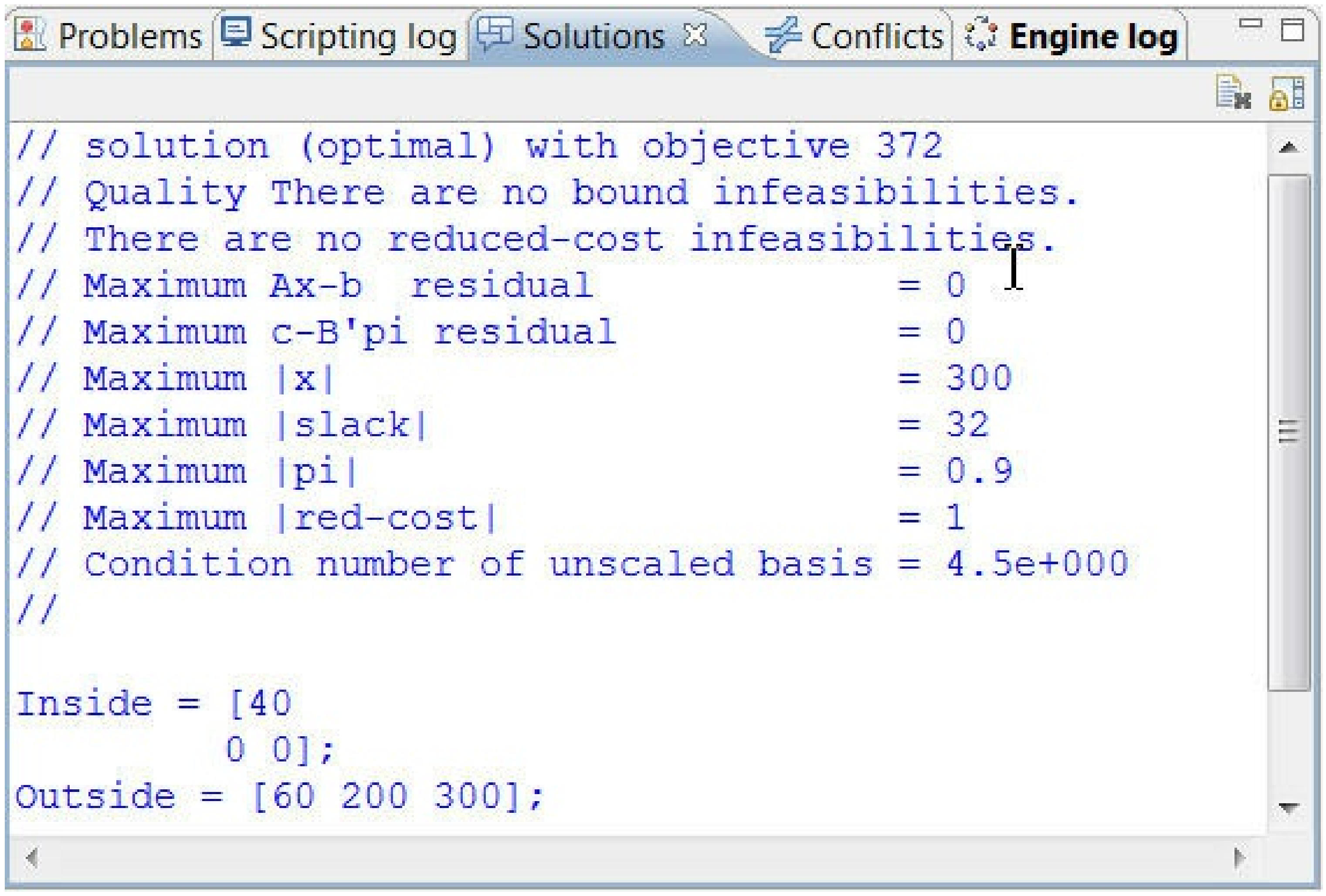


```
{string} Products = ...;
{string} Resources = ...;

float Consumption[Products][Resources] = ...;
float Capacity[Resources] = ...;
float Demand[Products] = ...;
float InsideCost[Products] = ...;
float OutsideCost[Products]  = ...;

dvar float+ Inside[Products];
dvar float+ Outside[Products];

minimize
  sum( p in Products )
    ( InsideCost[p] * Inside[p] + OutsideCost[p]
```

6. Right-click on **Basic Configuration** and select **Run this**.



7. After a few seconds, the optimal solution is displayed in the **Solutions** tab at the bottom of the IDE.



```
// solution (optimal) with objective 372
// Quality There are no bound infeasibilities.
// There are no reduced-cost infeasibilities.
// Maximum Ax-b  residual              = 0
// Maximum c-B'pi residual             = 0
// Maximum |x|                         = 300
// Maximum |slack|                     = 32
// Maximum |pi|                        = 0.9
// Maximum |red-cost|                  = 1
// Condition number of unscaled basis = 4.5e+000
//

Inside = [40
         0 0];
Outside = [60 200 300];
```

Parent topic:

→ Completely new to CPLEX Optimization Studio?