Changing the rules of business™

ILOG OPL Development Studio Getting Started with the OPL IDE

Copyright

General Use Restrictions

Trademarks

Open Source Software

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

Getting Started with the OPL IDE

This manual starts with an *Introduction to the OPL IDE* section; read it first. It is followed by a *Getting Started tutorial* in which you launch the OPL IDE, create an empty project, enter a model, add data and a settings file, create run configurations and execute the final project.

In this section

Introduction to the OPL IDE

Provides an overview of important concepts and features of the OPL IDE that you should be familiar with before starting to work with it in *Getting Started with the OPL IDE*.

In this section

Launching the OPL IDE
How to launch the OPL IDE, and descriptions of the Welcome window and the OPL main window.

Opening projects in the OPL Projects Navigator
How to open OPL projects and work with them using the OPL Projects Navigator.

Important concepts and terms
Describes some of the terms and concepts that are important to understand about the OPL IDE.

The Problem Browser
Describes the features of the OPL Problem Browser.

Resizing, moving, and hiding IDE views
Shows how to customize the appearance of the OPL IDE.

Working with files in OPL
Shows how to open, edit, and work with files in the OPL IDE.

Launching the OPL IDE

How to launch the OPL IDE, and descriptions of the Welcome window and the OPL main window.

In this section

Starting the IDE
Starting OPL IDE from Windows.

The Welcome window
Describes the OPL Welcome screen and how to close it and begin working with OPL.

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.

Starting the IDE

You can start the IDE from the Windows Start menu, from Windows Explorer, or from the command line.

To launch the IDE from the Start Menu:

1. Click the Windows **Start** menu.
2. Select **Programs > ILOG > ILOG OPL Development Studio [version_number] > OPL IDE**

To launch the IDE from Windows Explorer:

♦ Double-click the IDE executable `oplide.exe` in the `<OPL_dir>\oplide` directory, where `<OPL_dir>` is your installation directory.

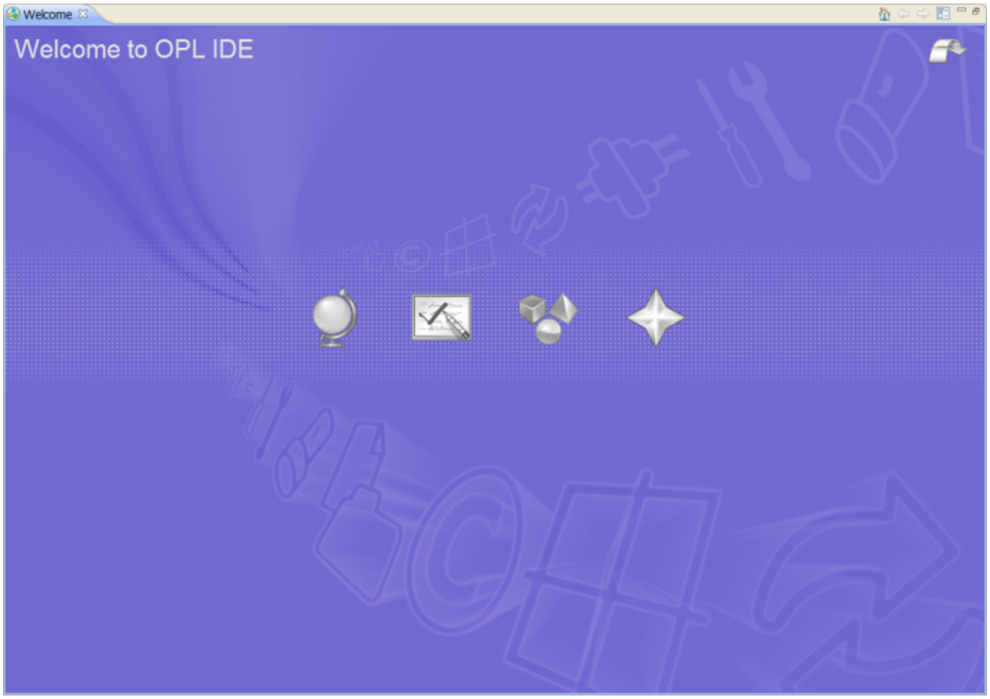
To launch the IDE from the command line:

1. Open a command prompt window.
2. Enter:

```
oplide
```

The Welcome window

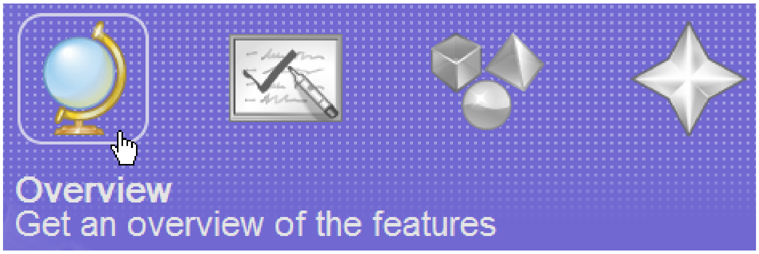
When you first launch the OPL IDE, a Welcome window displays:



The Welcome window presents access to what's new in the release, to the Samples provided in the distribution, and to other information.

To access the information on the Welcome window:

- Mouseover the buttons to see a tooltip that explains them:



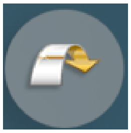
- Click the buttons to see the information.

The buttons contained on the Welcome window lead to information on the release and/or parts of the OPL documentation you might want to refer to often:

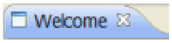
- **Overview** — this button displays an Overview page that contains:
  - A link to *A quick start to OPL* section of the documentation.
  - A link to the *Migrating from previous versions of OPL* section of the documentation.
- **Tutorials** — this button displays a page that contains a set of links to different sections of the *IDE Tutorials* manual.
- **Samples** — this button displays a page that contains a set of links to different sections of the *Language and Interfaces Examples* manual.
- **What's New** — this button displays an page that contains a set of links to:
  - The *New and changed in OPL* section of the *Release Notes*.
  - The *Introduction to the OPL IDE*, which provides an overview of new and changed features in the IDE.
  - A set of links to various OPL and ODM user forums. These links are driven by an RSS feed, so they are constantly updated to reflect the latest information on those forums.

To close the Welcome window and use the OPL IDE:

- Click the Workbench icon at the top right of the Welcome window:



- **OR**, click the **X** in the Welcome window tab to close it.



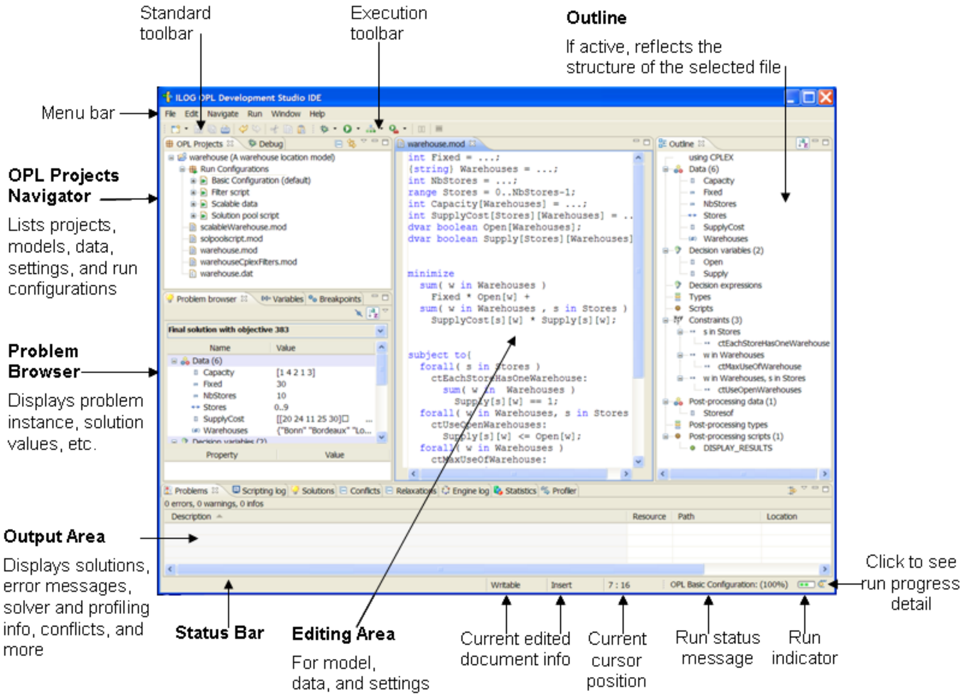
- When you close the Welcome window using either method, the OPL Main window appears. It is described in the next section.

To return to the Welcome window from the OPL IDE:

- Choose **Help>Welcome** from the main menu.

The Main window

The following illustration details the primary areas and controls of the OPL Main window. Tooltips appear when you move the pointer over most elements of the main window.



Name	Value
Data (6)	
Capacity	[1 4 2 1 3]
Fixed	30
NbStores	10
Stores	0..9
SupplyCost	[[20 24 11 25 30]□ ...
Warehouses	{"Bonn" "Bordeaux" "Lo...
Decision variables (2)	

```
int Fixed = ...;
{string} Warehouses = ...;
int NbStores = ...;
range Stores = 0..NbStores-1;
int Capacity[Warehouses] = ...;
int SupplyCost[Stores][Warehouses] = ..
dvar boolean Open[Warehouses];
dvar boolean Supply[Stores][Warehouses]


minimize
  sum( w in Warehouses )
    Fixed * Open[w] +
  sum( w in Warehouses , s in Stores )
    SupplyCost[s][w] * Supply[s][w];


subject to{
  forall( s in Stores )
    ctEachStoreHasOneWarehouse:
      sum( w in  Warehouses )
        Supply[s][w] == 1;
  forall( w in Warehouses, s in Stores
    ctUseOpenWarehouses:
      Supply[s][w] <= Open[w];
  forall( w in Warehouses )
    ctMaxUseOfWarehouse:
```

Standard toolbar

Execution toolbar

Outline

If active, reflects the structure of the selected file

Menu bar

OPL Projects Navigator

Lists projects, models, data, settings, and run configurations

Problem Browser

Displays problem instance, solution values, etc.

Output Area

Displays solutions, error messages, solver and profiling info, conflicts, and more

Status Bar

Editing Area

For model, data, and settings

Current edited document info

Current cursor position

Run status message

Run indicator

Click to see run progress detail

Opening projects in the OPL Projects Navigator

How to open OPL projects and work with them using the OPL Projects Navigator.

In this section

Importing existing projects into the workspace
Shows how to load OPL projects into the OPL IDE.

Managing projects in the OPL Projects Navigator
Explains how to work with your projects once you have imported them.

Importing existing projects into the workspace

Use the following procedure to import an existing OPL project into the OPL Projects Navigator.

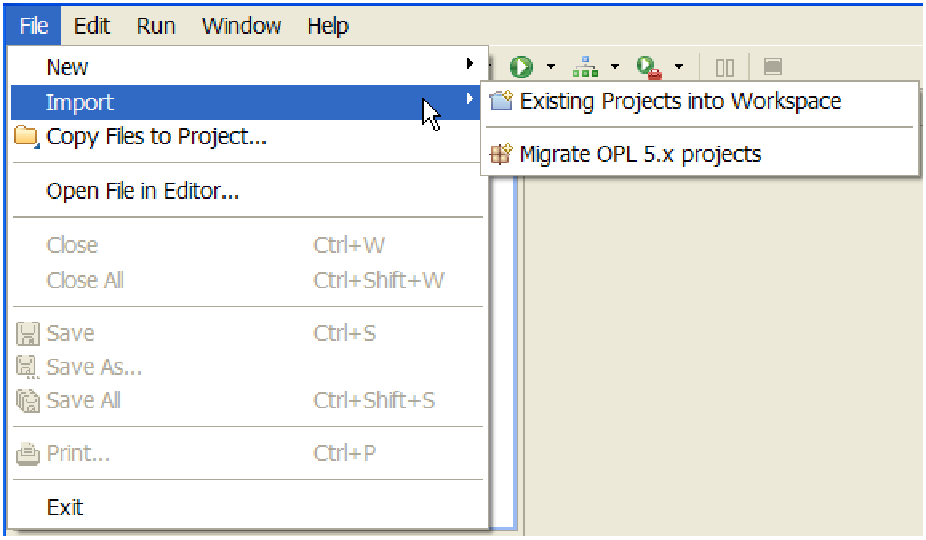
Note: This procedure is only for OPL projects. For importing projects from previous releases of OPL, see *Migrating from previous versions of OPL*.

In addition to the procedures detailed below, you can double-click on the files contained in an OPL project folder to open that project in the OPL IDE:

- ♦ If the project has already been added to the OPL Projects Navigator, double-clicking any model (`.mod`), data (`.dat`), or settings (`.ops`) file will open that project in the OPL IDE.
- ♦ If the project has **not** already been added to the OPL Projects Navigator, double-clicking on any of these files will launch the Import Wizard discussed in the following procedures.

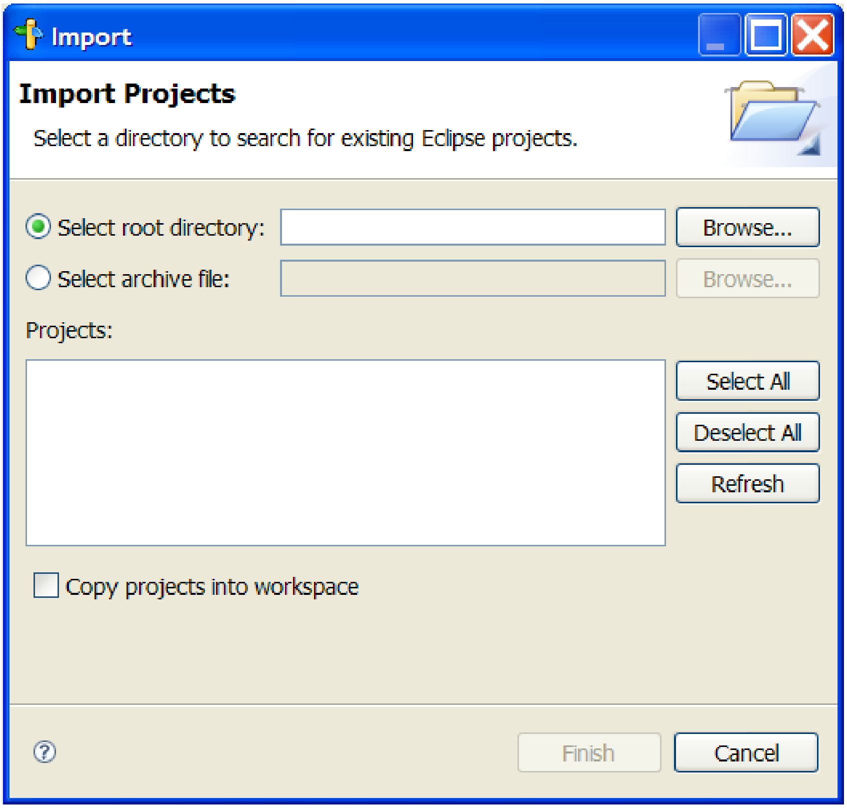
With the OPL IDE running:

1. From the main menu, choose **File>Import**, or right-click in the OPL Projects Navigator and choose **Import**.



2. Select **Existing Projects Into Workspace**.

   The first screen of the Import Wizard is displayed:



Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory:

○ Select archive file:

□ Copy projects into workspace

This screen can be used to load one or more OPL projects into the OPL Projects Navigator. The general procedure for doing this is to:

- Select a root directory (this is the directory on your file system where your existing OPL project is located). Alternatively you can select an archive file where the project has been stored.
- Select a project (or projects, if the root directory contains more than one project) to be imported.
- Indicate whether you want to copy the projects into the workspace.

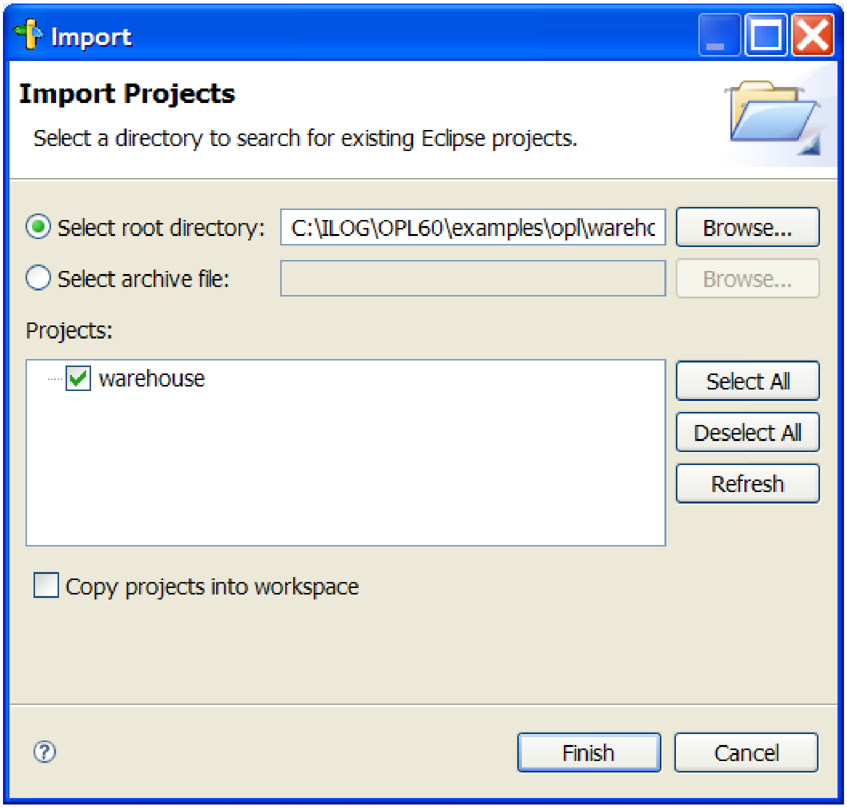
The next steps walk you through each of the general procedures above, using the example of importing the `warehouse` example from the OPL distribution.

3. In the **Select root directory** field, enter the pathname of the directory that contains the project(s) you want to import. You can type in the pathname or use the **Browse** button to search for it.

Note: Alternatively, you can use the **Select archive file** field and enter the pathname of a JAR, ZIP, TAR or other compressed file that contains your project(s).

After you have selected a root directory (or archive file), the OPL projects under that directory that do not currently exist in your workspace are listed in the **Projects** view.

4. Check the box of each of the projects you want to import. (In the example used in this procedure, there is only one, `warehouse`.)



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: C:\ILOG\OPL60\examples\opl\warehc

○ Select archive file:

Projects:

☑ warehouse

□ Copy projects into workspace

5. Leave the **Copy projects into workspace** box unchecked if you want to work with the project “in place” in its current location, or check the box to copy it to your workspace, and click **Finish** to import the project(s) into your OPL Projects Navigator.

Managing projects in the OPL Projects Navigator

After you have migrated OPL 5.x projects or imported or created new OPL 6.x projects, you can leave them in your OPL Projects Navigator. If you exit from OPL, when you next launch the OPL IDE, they will be there, ready to use.

If you have loaded a number of projects and the OPL Projects Navigator starts to get “crowded,” there are two ways to save memory or space in the Navigator window — by closing (collapsing) the projects or by deleting them.

Closing/Opening projects

Projects are either open or closed. When a project is closed, it cannot be changed, but its resources still reside on the local file system. Because they are not examined during builds, closed projects require less memory. Therefore, closing projects you are not working with can improve build time.

♦ Right-click on the project name and choose **Close project** from the context menu to close the project.

The plus sign next to the project name disappears, but it remains in the OPL Projects Navigator.

♦ To reopen the project, right-click on the project name and choose **Open project** from the context menu.

Deleting projects

If you are not currently working with a project, you can also safely delete it from the OPL Projects Navigator, without deleting it from the file system.

♦ To remove a project from the OPL Projects Navigator, right-click on the project name and choose **Delete** from the context menu.

A popup message appears asking whether you want to delete the project only from the workspace, or from the file system as well.

The two options available to you are:

- Select **Also delete contents...** to delete the project entirely.

  This means that the project will be completely deleted, and cannot later be recovered using **Undo** or the **Import>Existing Projects Into Workspace** menu command.

- Select **Do not delete contents** (the default) to remove the project from the OPL Projects Navigator but leave it on the file system.

  This means that the project is still present on the file system and can be reopened using the **Import>Existing Projects Into Workspace** menu command.

Important concepts and terms

Describes some of the terms and concepts that are important to understand about the OPL IDE.

In this section

Resources

Resources refers to the projects, folders, and files that exist in the OPL IDE. The OPL Projects Navigator provides a hierarchical view of these resources and allows you to open them for editing. There are three basic types of resources:

- **Files** — Similar to files in the file system.
- **Folders** — Similar to folders in the file system.
- **Projects** — Used for builds, version management, sharing, and resource organization. Projects are comprised of folders and files, and map to directories in the file system.

Workspace

The *workspace* is the root directory in which you store and work with your resources. It can be located anywhere on the file system, but its default location is `C:\Documents and Settings\<user>\Application Data\ILOG\OPL Studio IDE\6.x\`.

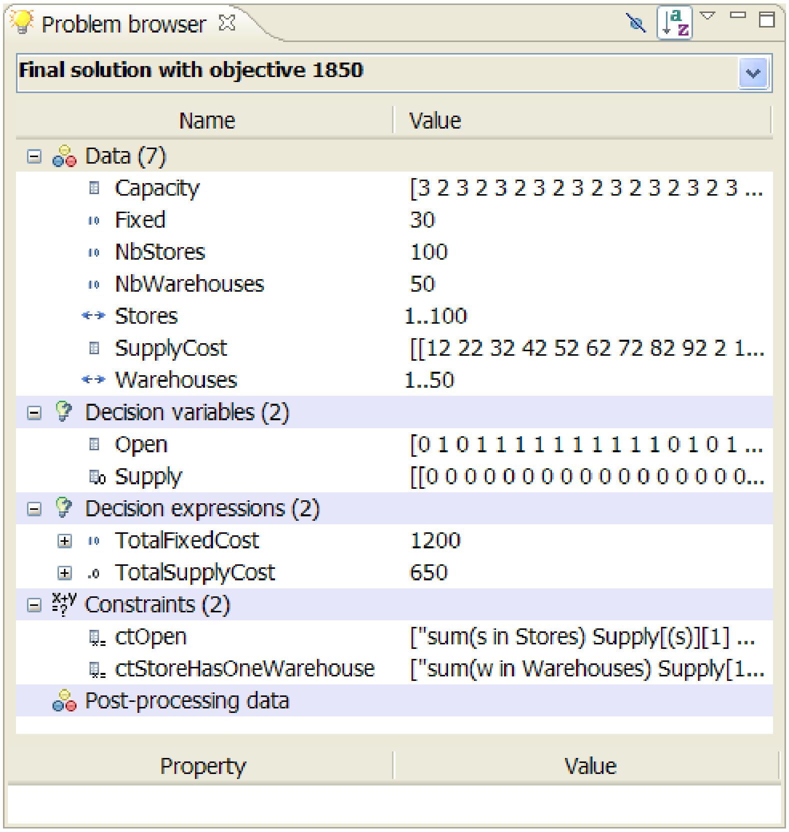
Views

Views is the name for the various panes or windows within the OPL IDE frame. Views can be editors or navigators or provide alternative ways to visualize and work with your projects. For example, the OPL Projects Navigator view displays the resources in your OPL projects, and allows you to open them in editor views.

Views may also have their own menus and some views have their own toolbars.

The Problem Browser

The Problem Browser in OPL 6.x works similarly to the Problem Browser in OPL 5.x.



Problem browser

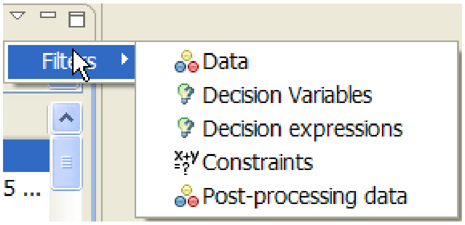
Final solution with objective 1850

Name	Value
Data (7)	
Capacity	[3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 ...
Fixed	30
NbStores	100
NbWarehouses	50
Stores	1..100
SupplyCost	[[12 22 32 42 52 62 72 82 92 2 1...
Warehouses	1..50
Decision variables (2)	
Open	[0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 ...
Supply	[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0...
Decision expressions (2)	
TotalFixedCost	1200
TotalSupplyCost	650
Constraints (2)	
ctOpen	["sum(s in Stores) Supply[(s)][1] ...
ctStoreHasOneWarehouse	["sum(w in Warehouses) Supply[1...
Post-processing data	

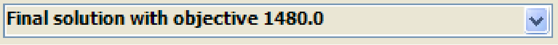
Property	Value

Some of the features of the Problem Browser are described briefly below:

- Hide properties button to conserve space.
- Sort button to sort .
- Filters menu allows filtering of displayed element types:



- Final solution status is displayed in the drop-down list at the top of the view:


Final solution with objective 1480.0

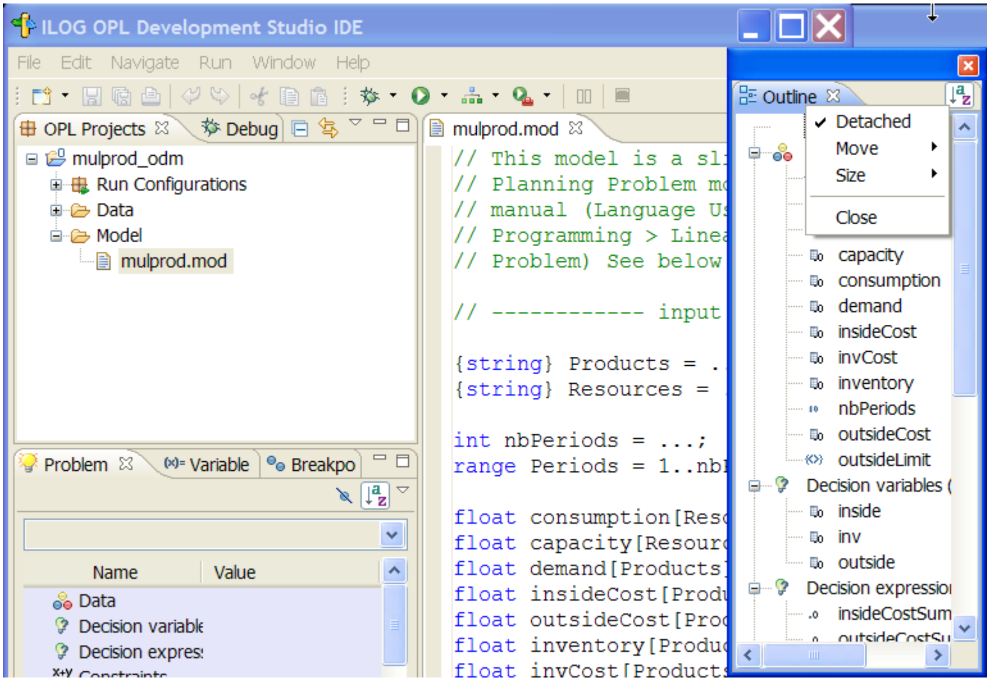
  When solving MIPS problems, multiple solutions may be displayed in this drop-down list. Choosing one of them displays data for that solution in the lower part of the Problem Browser.

- Tooltips show data that is too wide to display.
- Double-clicking on an item opens an editor for that item.
- For Decision Variables, some information formerly shown in properties view has been moved to the editor view itself, so one can safely turn properties view off.

Resizing, moving, and hiding IDE views

All views are resizable, movable, dockable, and detachable. *Dockable* means that you can drag a view from one location and drop it in another, even within another view. For example, you could drag the Outline view into the Output Area, and it would become another tab there.

Detachable means that you can drag a view outside the OPL IDE frame and it becomes its own standalone window. When you do this, to put the view back into the frame you need to right-click in the tab area of the detached view and uncheck the **Detached** item in the context menu:



```
// This model is a sl
// Planning Problem m
// manual (Language U
// Programming > Linea
// Problem) See below

// ------------ input

{string} Products = .
{string} Resources =

int nbPeriods = ...;
range Periods = 1..nb

float consumption[Res
float capacity[Resour
float demand[Products]
float insideCost[Prod
float outsideCost[Pro
float inventory[Produ
float invCost[Product
```

When you do this, the detached view will return to the frame.

Resizing views

To resize views, click on one of its borders or corners and drag it to the desired size.

Or, to temporarily expand a view to the full size of the IDE frame, double-click its tab. Double-click again to shrink it back to original size.

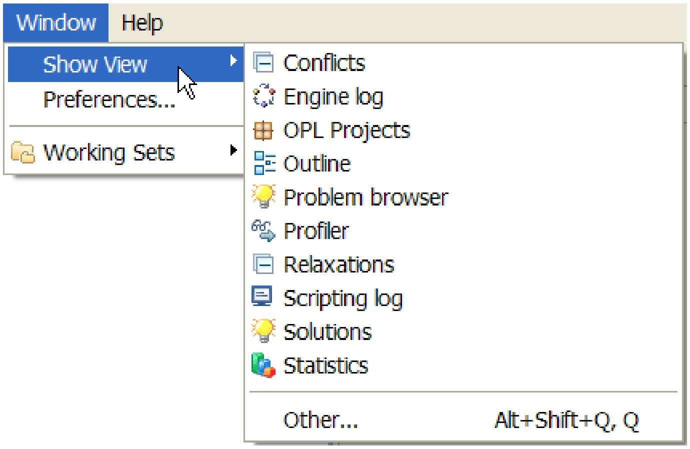
Moving views

To move a view, right-click its top border and drag it to the desired location. A black border appears as you drag the view around the frame, to tell you where the view will be placed if you release the mouse button in that location.

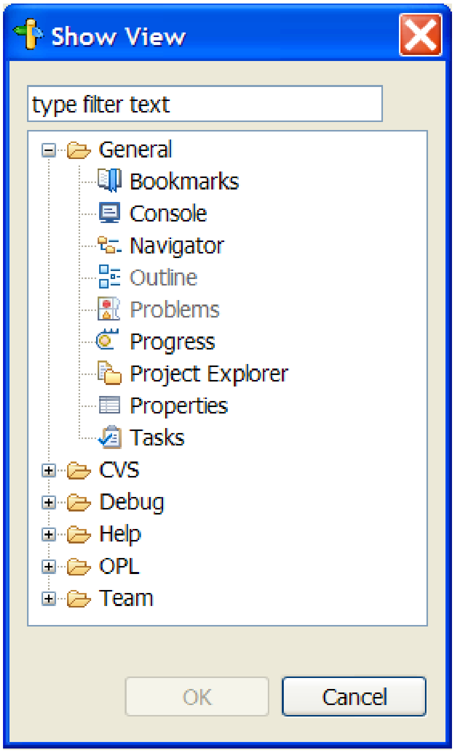
Hiding views

To hide a view, click its Close box ☒.

To display a view that has been closed, in the main menu choose **Window>Show View** and select the name of the view you want to display from the menu:



If the view you want to display is not shown in that menu, click Other to display a popup window that contains more views:


Show View
type filter text
General
Bookmarks
Console
Navigator
Outline
Problems
Progress
Project Explorer
Properties
Tasks
CVS
Debug
Help
OPL
Team
OK
Cancel

Working with files in OPL

Shows how to open, edit, and work with files in the OPL IDE.

In this section

Adding existing files to a project

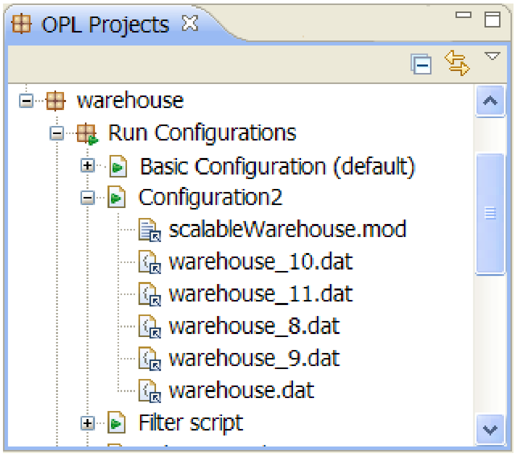
You can use the **File>Copy Files to Project** command to open a dialog box that allows you to open files and import them into selected projects.

You can also drag existing files from a Windows Explorer window and drop them onto the project folder in your OPL Projects Navigator. This is always a *copy* operation, not a move.

Ordering files within a run configuration

When you execute a run configuration, the order of the data or settings files relative to each other is important. Since some data in a `.dat` file may depend on other data in a different `.dat` file, if the data files are in the wrong order it may cause an error at execution time.

As an example of this, look at the following screenshot of a run configuration:



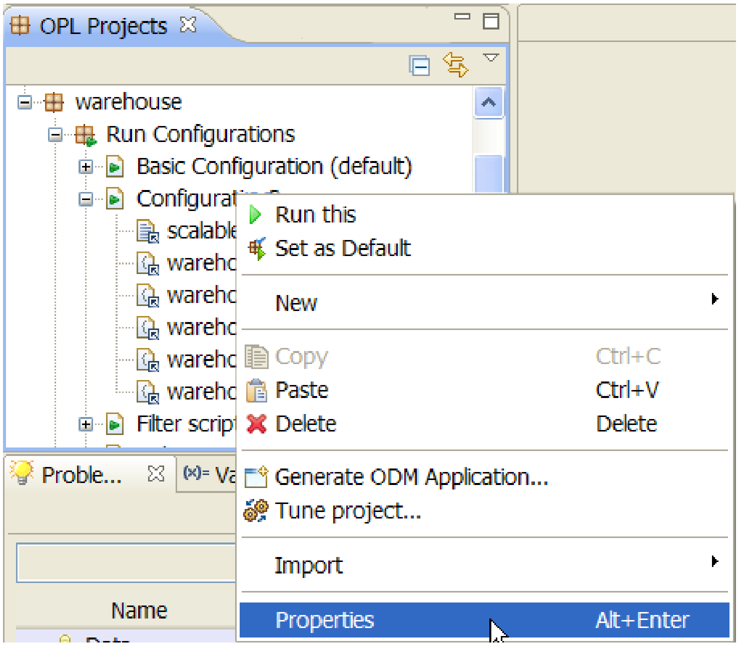
```
OPL Projects
warehouse
  Run Configurations
    Basic Configuration (default)
    Configuration2
      scalableWarehouse.mod
      warehouse_10.dat
      warehouse_11.dat
      warehouse_8.dat
      warehouse_9.dat
      warehouse.dat
    Filter script
```

In this run configuration, the four data files are intended to be executed in numerical order. However, as you can see, they are sorted in ASCII order. This would cause them to be executed in the wrong sequence.

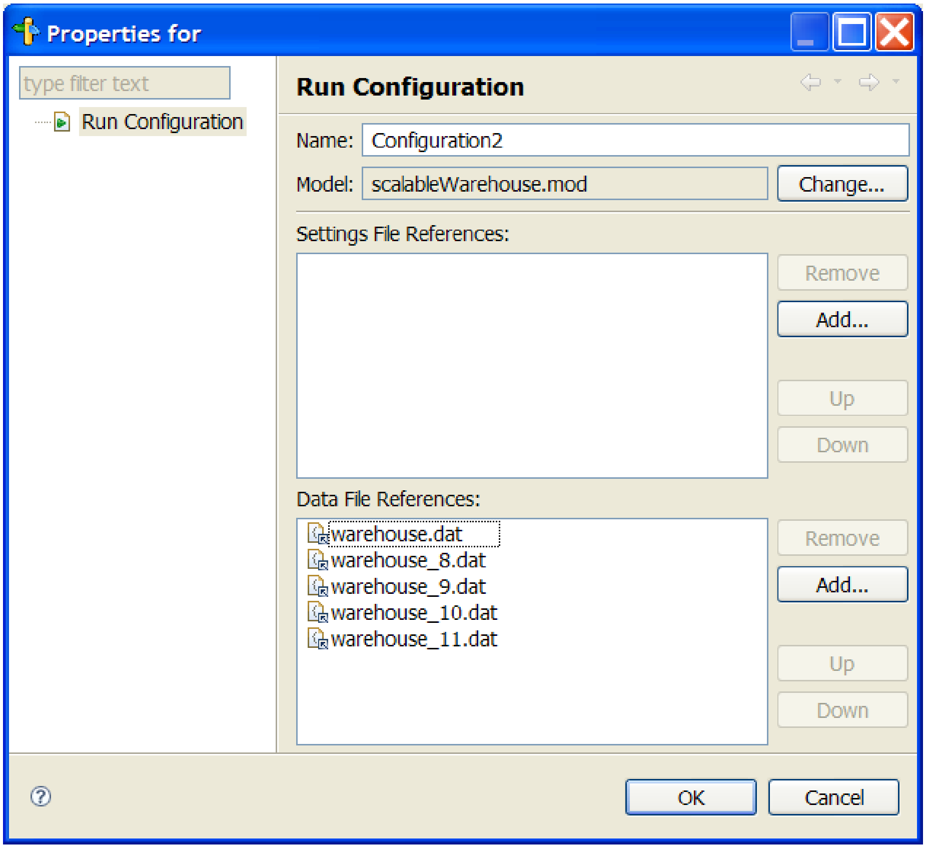
You can set the order of multiple data or settings files in a run configuration using the following procedure.

To set the order of files within a run configuration:

1. Right-click on the run configuration name and choose **Properties** from the context menu:



2. A properties window appears for the run configuration:



Name: Configuration2

Model: scalableWarehouse.mod

Data File References:

- warehouse.dat
- warehouse_8.dat
- warehouse_9.dat
- warehouse_10.dat
- warehouse_11.dat

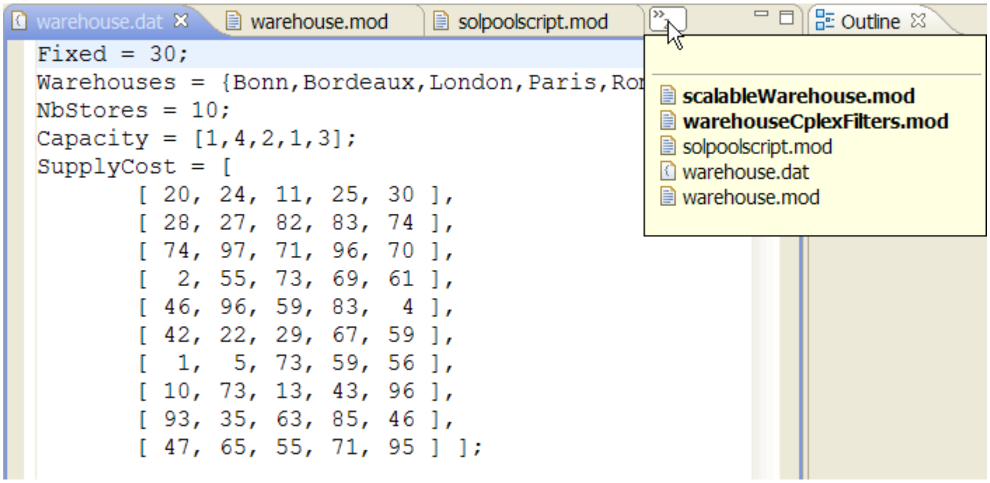
You can use the **Up** and **Down** buttons to rearrange the order of your settings files and data files. In the example above, the data files have been reset to the proper order.

Opening files for editing

In general, you open your model files, data files and settings files for editing by double-clicking on the file in OPL Projects Navigator.

Multiple editors can be open in the Editing Area at once. You can switch back and forth between the open views by clicking the tabs of the views that are visible in the Editing Area.

If so many edit views are open that all of their tabs cannot be displayed, a “more views” icon »2 becomes visible. Click it and a list of the other views appears:



```
Fixed = 30;
Warehouses = {Bonn,Bordeaux,London,Paris,Ro
NbStores = 10;
Capacity = [1,4,2,1,3];
SupplyCost = [
         [ 20, 24, 11, 25, 30 ],
         [ 28, 27, 82, 83, 74 ],
         [ 74, 97, 71, 96, 70 ],
         [  2, 55, 73, 69, 61 ],
         [ 46, 96, 59, 83,  4 ],
         [ 42, 22, 29, 67, 59 ],
         [  1,  5, 73, 59, 56 ],
         [ 10, 73, 13, 43, 96 ],
         [ 93, 35, 63, 85, 46 ],
         [ 47, 65, 55, 71, 95 ] ];
```

Click any of the views in the popup list to view them.

Opening external files in the Editor

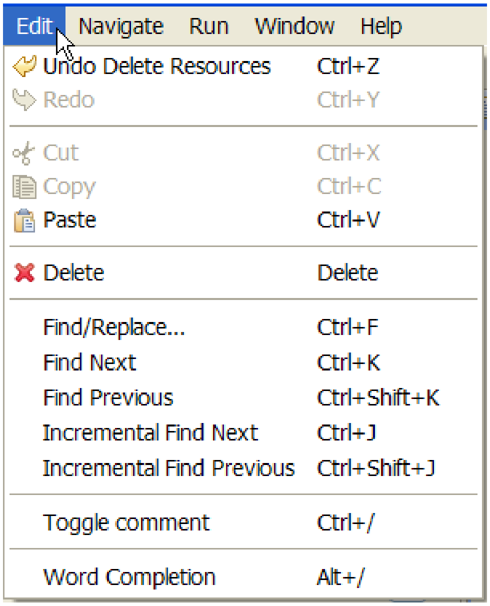
To open files that are not currently in any open project in OPL Projects Navigator, use the **File>Open File in Editor** menu command.

This does not add the files to any project, but allows you to view or edit them in the Editing Area. To add external files to a project, see the *Adding existing files to a project* section.

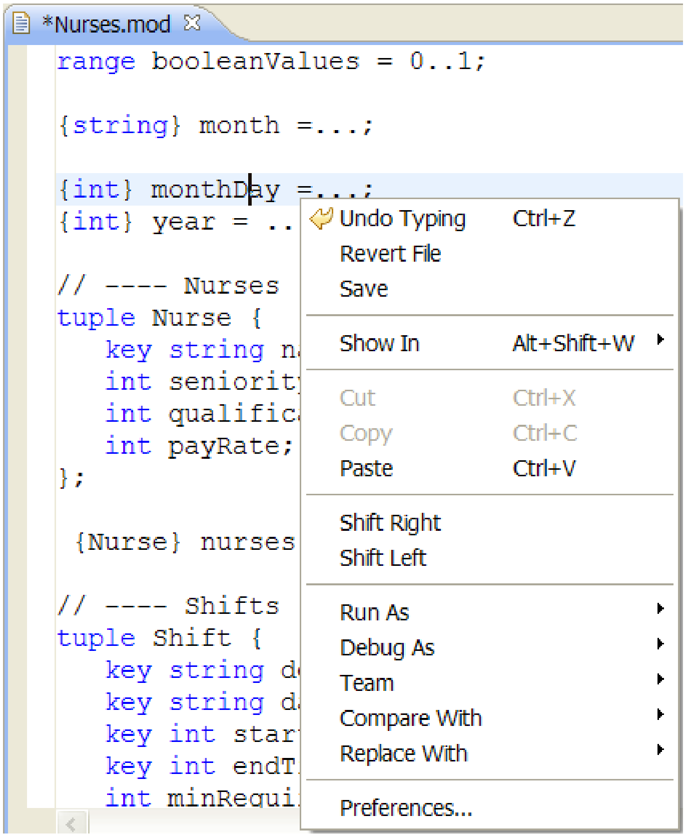
Editing commands and shortcuts

The common commands you need to use while editing can be found on two menus:

- The main menu **Edit** menu:



♦ The right-click context menu within the editor itself:



```
range booleanValues = 0..1;

{string} month =...;

{int} monthDay =...;
{int} year = ..

// ---- Nurses
tuple Nurse {
   key string n
   int seniorit
   int qualific
   int payRate;
};

 {Nurse} nurses

// ---- Shifts
tuple Shift {
   key string d
   key string d
   key int star
   key int endT
   int minRequi
```

Local History and its related features

OPL 6.x provides a limited form of version control called **Local History** that allows you to track and compare different versions of your files as you edit them over the lifespan of a project.

For example, if you edit the same model file several times, all versions of the file are still available to you. You can use the **Compare With** and **Replace With** commands to compare different versions of a file or revert to previous versions of a file or its contents.

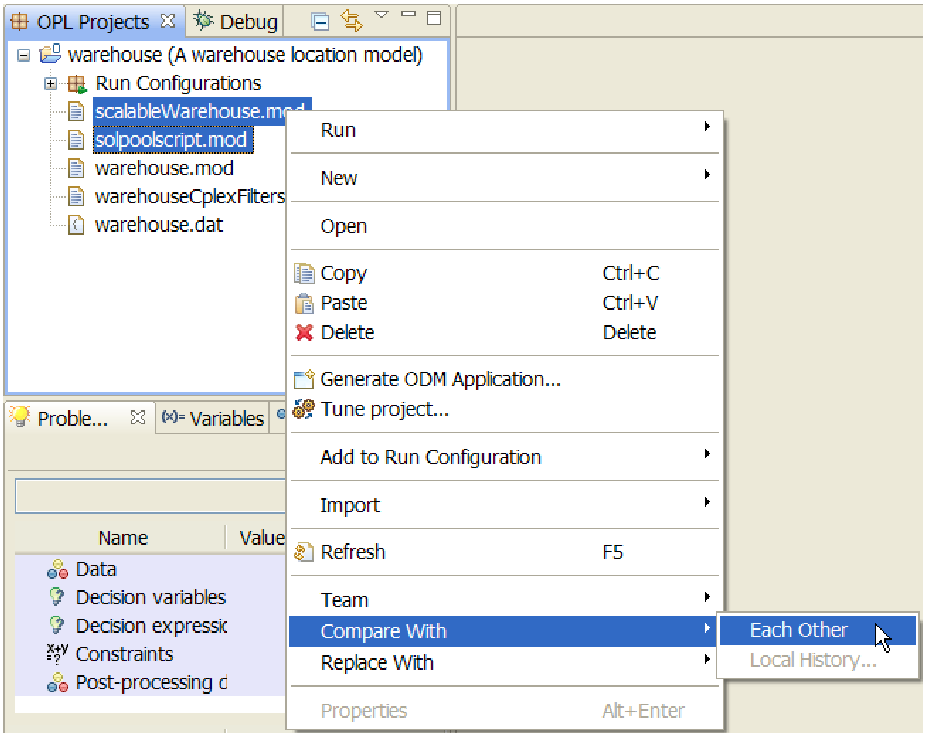
Each of these features is described in the following sections

'Compare With' features

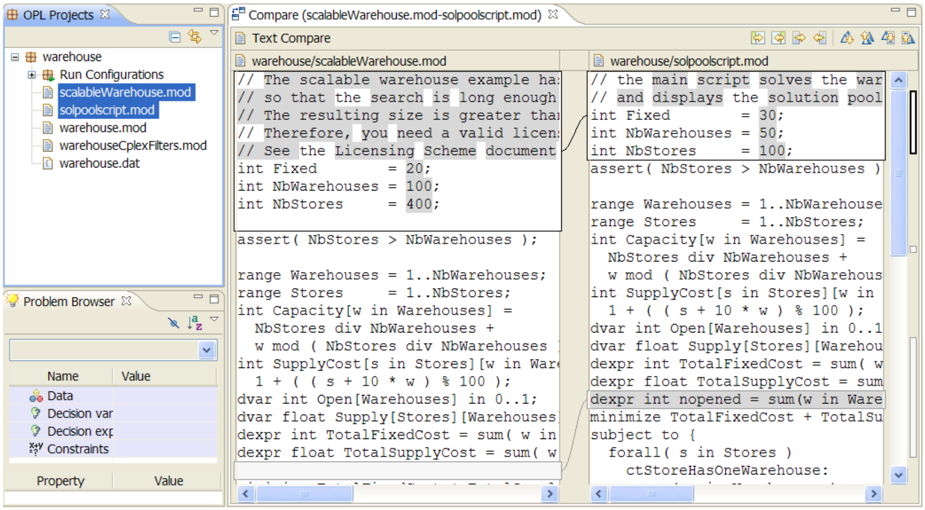
Compare With Each Other

To compare two files in the same project with each other:

1. In the OPL Projects Navigator, highlight the two files you want to compare and right-click to display a context menu and choose **Compare With>Each Other** from that menu:



2. The files are opened in the Editing Area in a special view that allows you to view them side by side, with the differences between the two files highlighted:



warehouse/scalableWarehouse.mod

```
// The scalable warehouse example ha
// so that the search is long enough
// The resulting size is greater tha
// Therefore, you need a valid licen
// See the Licensing Scheme document
int Fixed         = 20;
int NbWarehouses = 100;
int NbStores      = 400;

assert( NbStores > NbWarehouses );

range Warehouses = 1..NbWarehouses;
range Stores      = 1..NbStores;
int Capacity[w in Warehouses] =
  NbStores div NbWarehouses +
  w mod ( NbStores div NbWarehouses
int SupplyCost[s in Stores][w in War
  1 + ( ( s + 10 * w ) % 100 );
dvar int Open[Warehouses] in 0..1;
dvar float Supply[Stores][Warehouses
dexpr int TotalFixedCost = sum( w in
dexpr float TotalSupplyCost = sum( w
```

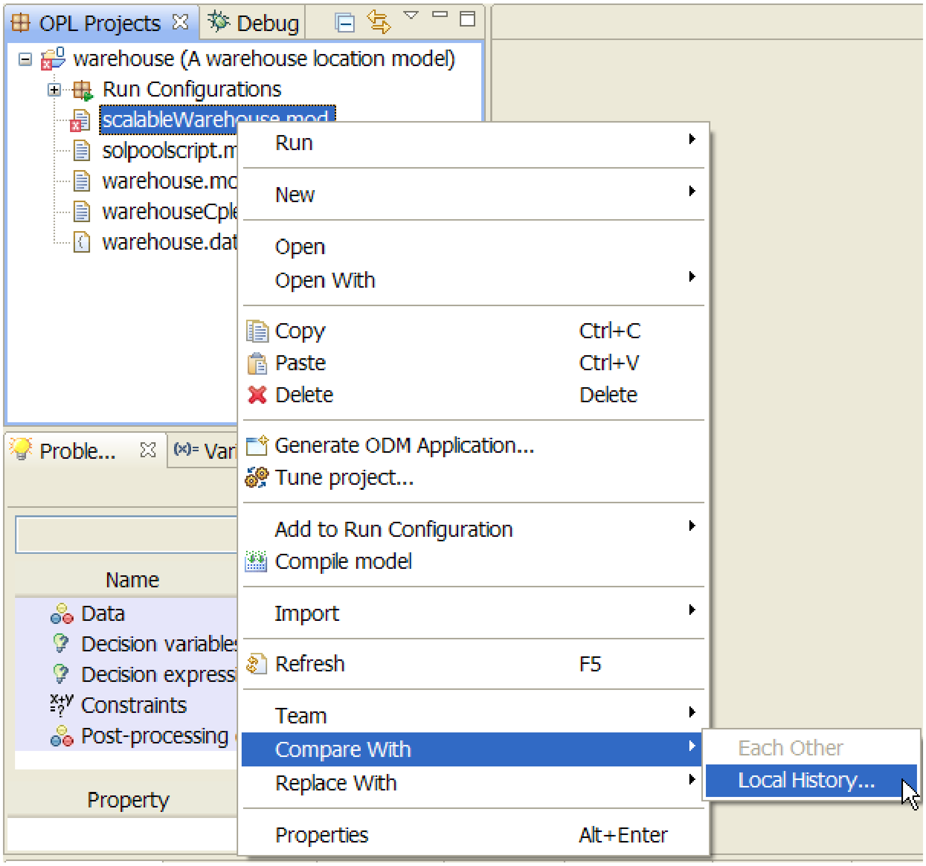
warehouse/solpoolscript.mod

```
// the main script solves the war
// and displays the solution pool
int Fixed         = 30;
int NbWarehouses = 50;
int NbStores      = 100;
assert( NbStores > NbWarehouses )

range Warehouses = 1..NbWarehouse
range Stores      = 1..NbStores;
int Capacity[w in Warehouses] =
  NbStores div NbWarehouses +
  w mod ( NbStores div NbWarehous
int SupplyCost[s in Stores][w in
  1 + ( ( s + 10 * w ) % 100 );
dvar int Open[Warehouses] in 0..1
dvar float Supply[Stores][Warehou
dexpr int TotalFixedCost = sum( w
dexpr float TotalSupplyCost = sum
dexpr int nopened = sum(w in Ware
minimize TotalFixedCost + TotalSu
subject to {
  forall( s in Stores )
    ctStoreHasOneWarehouse:
```

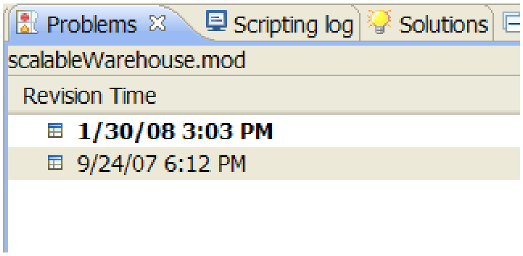
Compare With Local History

To compare a file with another version of itself in Local History:

1. In the OPL Projects Navigator, highlight the file you want to compare to its own Local History versions and right-click to display a context menu. Then choose **Compare With>Local History** from that menu:



2. In the Output Area of the OPL IDE, a list appears of the different versions of this file in Local History:

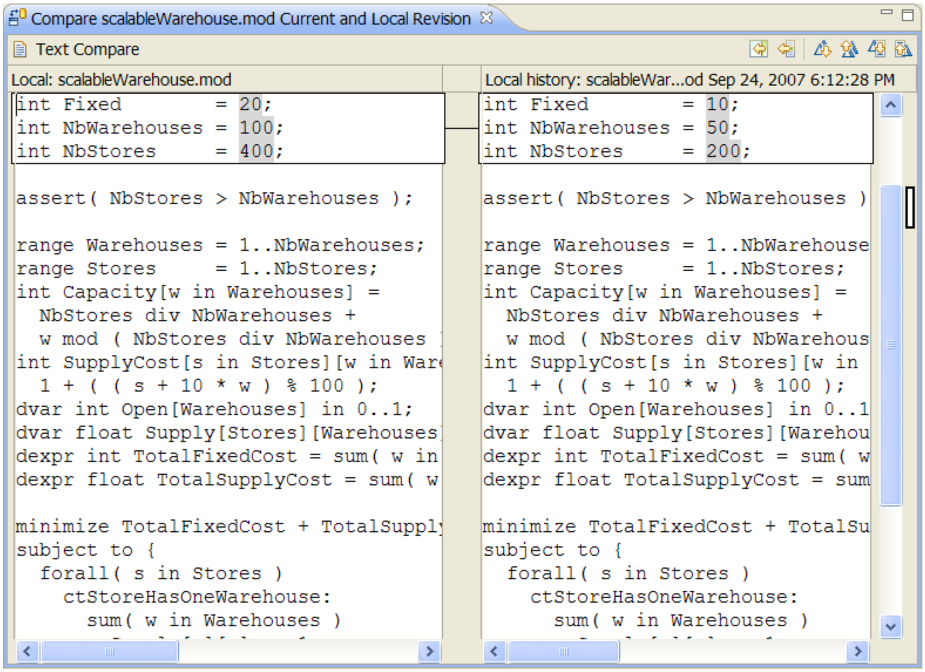


scalableWarehouse.mod

Revision Time
1/30/08 3:03 PM
9/24/07 6:12 PM

3. Double-click on the version of the file that you want to compare to the current version.

The files are opened in the Editing Area in a special view that allows you to view them side by side, with the differences between the two files highlighted:



Local: scalableWarehouse.mod

```
int Fixed         = 20;
int NbWarehouses = 100;
int NbStores      = 400;

assert( NbStores > NbWarehouses );

range Warehouses = 1..NbWarehouses;
range Stores      = 1..NbStores;
int Capacity[w in Warehouses] =
  NbStores div NbWarehouses +
  w mod ( NbStores div NbWarehouses )
int SupplyCost[s in Stores][w in Ware
  1 + ( ( s + 10 * w ) % 100 );
dvar int Open[Warehouses] in 0..1;
dvar float Supply[Stores][Warehouses]
dexpr int TotalFixedCost = sum( w in
dexpr float TotalSupplyCost = sum( w

minimize TotalFixedCost + TotalSuppl
subject to {
  forall( s in Stores )
    ctStoreHasOneWarehouse:
      sum( w in Warehouses )
```

Local history: scalableWar...od Sep 24, 2007 6:12:28 PM

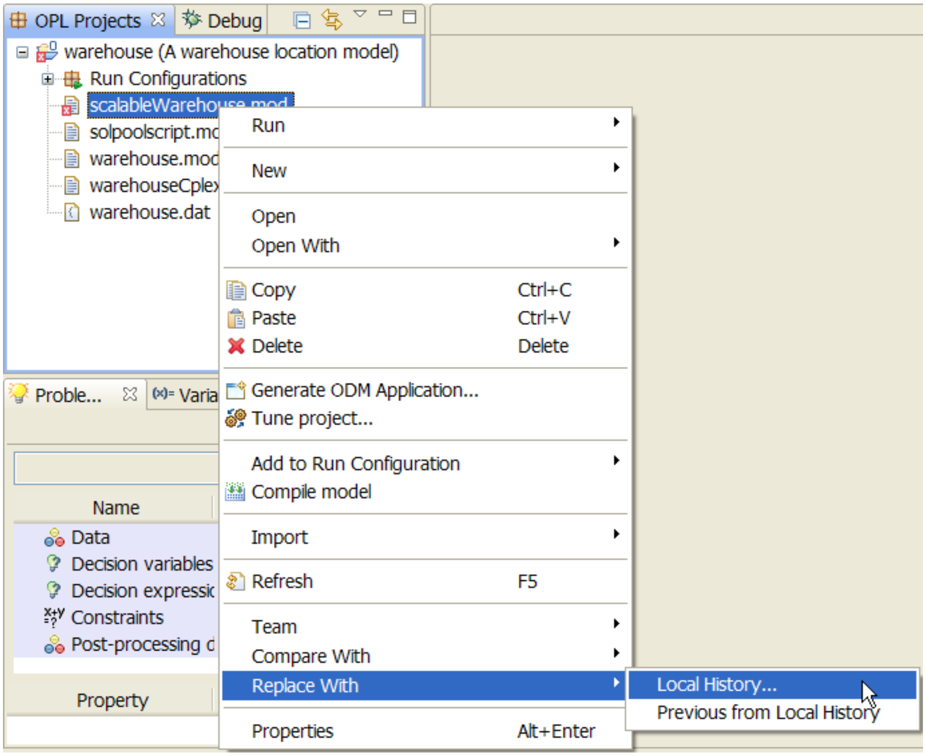
```
int Fixed         = 10;
int NbWarehouses = 50;
int NbStores      = 200;

assert( NbStores > NbWarehouses )

range Warehouses = 1..NbWarehouse
range Stores      = 1..NbStores;
int Capacity[w in Warehouses] =
  NbStores div NbWarehouses +
  w mod ( NbStores div NbWarehous
int SupplyCost[s in Stores][w in
  1 + ( ( s + 10 * w ) % 100 );
dvar int Open[Warehouses] in 0..1
dvar float Supply[Stores][Warehou
dexpr int TotalFixedCost = sum( w
dexpr float TotalSupplyCost = sum

minimize TotalFixedCost + TotalSu
subject to {
  forall( s in Stores )
    ctStoreHasOneWarehouse:
      sum( w in Warehouses )
```

'Replace With' features

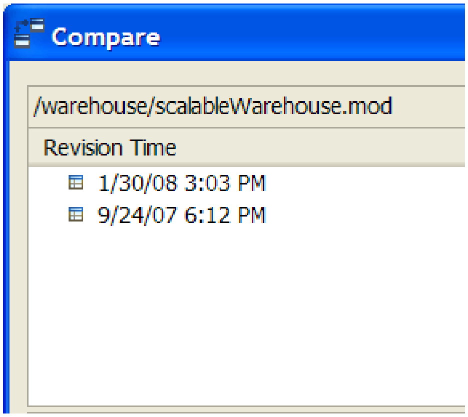
Replace With Local History

To replace a file with a selected version of itself from Local History:

1. In the OPL Projects Navigator, highlight the file you want to revert to one of its Local History versions and right-click to display a context menu. Then choose **Replace With>Local History** from that menu:

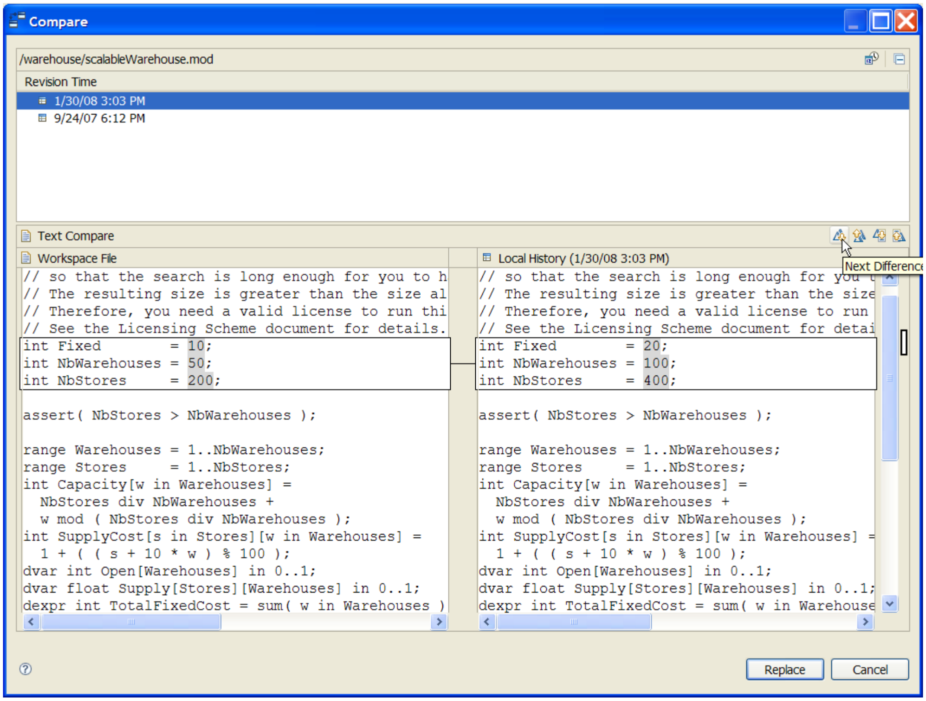


2. A popup window appears in which you can view the different versions of this file in Local History:



3. Double-click on the version of the file that you want to compare to the current version and use as a candidate for replacement.

   The files are opened in a special view that allows you to view them side by side, with the differences between the two files highlighted:



```
// so that the search is long enough for you to h
// The resulting size is greater than the size al
// Therefore, you need a valid license to run thi
// See the Licensing Scheme document for details.
int Fixed        = 10;
int NbWarehouses = 50;
int NbStores     = 200;

assert( NbStores > NbWarehouses );

range Warehouses = 1..NbWarehouses;
range Stores     = 1..NbStores;
int Capacity[w in Warehouses] =
  NbStores div NbWarehouses +
  w mod ( NbStores div NbWarehouses );
int SupplyCost[s in Stores][w in Warehouses] =
  1 + ( ( s + 10 * w ) % 100 );
dvar int Open[Warehouses] in 0..1;
dvar float Supply[Stores][Warehouses] in 0..1;
dexpr int TotalFixedCost = sum( w in Warehouses )
```

```
// so that the search is long enough for you t
// The resulting size is greater than the size
// Therefore, you need a valid license to run
// See the Licensing Scheme document for detai
int Fixed        = 20;
int NbWarehouses = 100;
int NbStores     = 400;

assert( NbStores > NbWarehouses );

range Warehouses = 1..NbWarehouses;
range Stores     = 1..NbStores;
int Capacity[w in Warehouses] =
  NbStores div NbWarehouses +
  w mod ( NbStores div NbWarehouses );
int SupplyCost[s in Stores][w in Warehouses] =
  1 + ( ( s + 10 * w ) % 100 );
dvar int Open[Warehouses] in 0..1;
dvar float Supply[Stores][Warehouses] in 0..1;
dexpr int TotalFixedCost = sum( w in Warehouse
```

You can use the icons that appear on the right of the lower window to move to the next and previous differences in the files, and thus determine whether you want to replace the contents of the current file with the contents of the version you are comparing it to.

Replace With Previous Version

To replace a file with its previous version in Local History:

1. In the OPL Projects Navigator, highlight the file you want to revert to its previous version and right-click to display a context menu. Choose **Replace With>Previous Version** from that menu.
2. The file is automatically replaced with the version of the file from Local History that immediately precedes it. This creates a new version in Local History.

Executing OPL projects

Describes the different ways of running and browsing OPL projects.

In this section

The Run options
Explains how to use the menus to run projects in the OPL IDE.

The Status Bar
Describes the area that displays messages about the current execution status of the IDE and information about files being edited.

The Run execution toolbar options
Explains how to use the menus to run projects in the OPL IDE.

The Run options

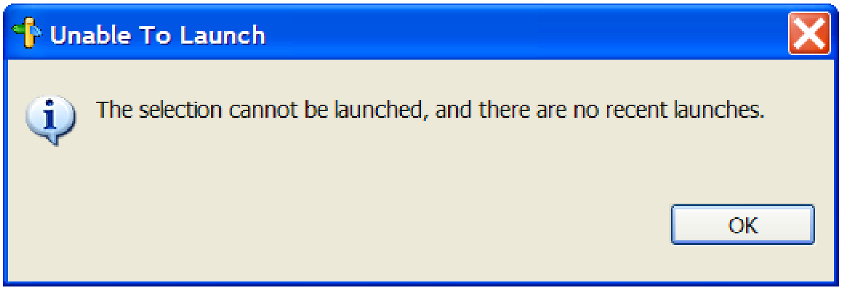
You can run your OPL projects by clicking the **Run** button in the execution toolbar.

You can also use right-click *context menus* to launch your projects.

How the Run button works

The behavior of the **Run** button in the execution toolbar depends on your “run history.”

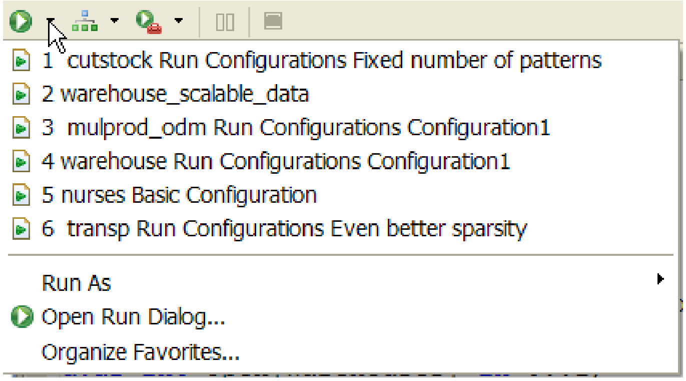
- If you have just launched OPL and no models been run yet, clicking the **Run** button may produce the following message:



Unable To Launch

The selection cannot be launched, and there are no recent launches.

- As runs are executed, they are added to a numbered list that is visible by clicking the arrow button to the right of the **Run** button.


1 cutstock Run Configurations Fixed number of patterns
2 warehouse_scalable_data
3 mulprod_odm Run Configurations Configuration1
4 warehouse Run Configurations Configuration1
5 nurses Basic Configuration
6 transp Run Configurations Even better sparsity
Run As
Open Run Dialog...
Organize Favorites...

- Once this list is populated, clicking the **Run** button launches *the most recently launched run configuration in the list*, no matter what project is selected in the OPL Projects Navigator.

Note: Obviously, this does not make it possible to just click a project in the OPL Projects Navigator and launch its default run configuration by simply clicking the **Run** button. For this reason, many OPL developers prefer the right-click context menus to launch their OPL models.

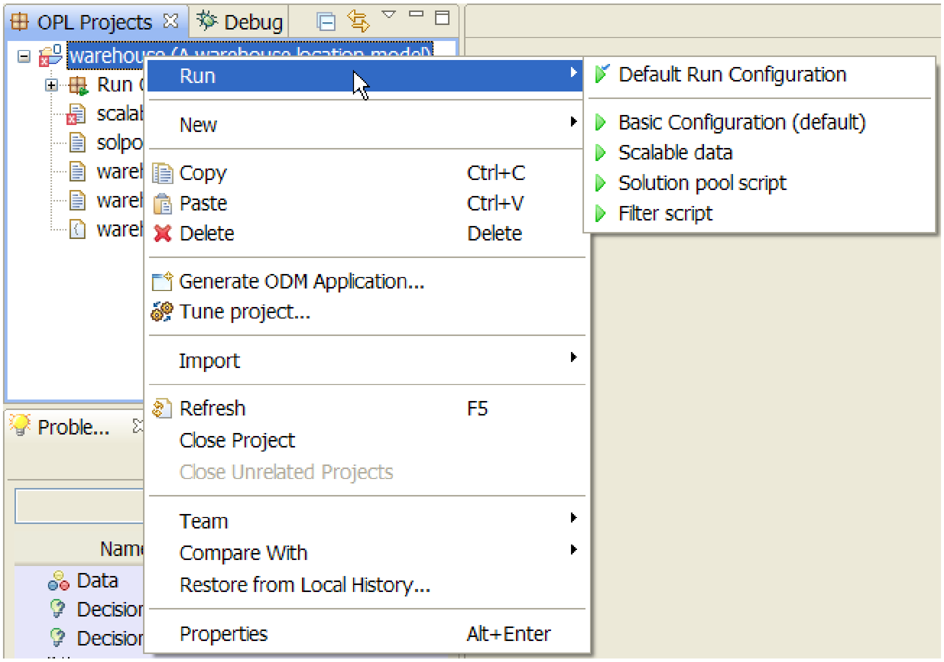
The default behavior of the **Run** button is configurable. See *Run, Debug, Browse, and External Tools button options* for more information.

How the Run context menus work

There are additional ways to run your projects in OPL. This section describes the context menu options that can be used to launch your projects directly from the OPL Projects Navigator.

To run your projects from the OPL Projects Navigator:

1. If you right-click on the project folder, or on any resource in the project folder **except** an individual run configuration you see the following context menu:

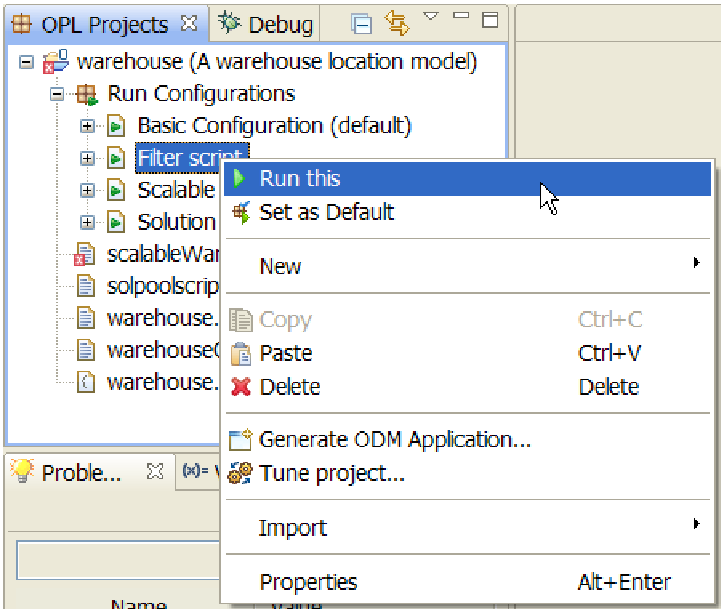


Two run options are listed:

- **Run** > **Default Run Configuration** — this option runs the run configuration that is currently set as the default for this project.

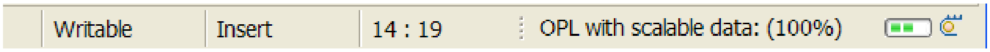
- **Run** > **<list_of_run_configurations>** — this option displays *all* run configurations for the project, so that you can choose which one you want to launch, whether it is currently the default run configuration or not.

2. **OR**, if you right-click on an individual run configuration for a project, you see a **Run this** menu option that enables you to run only that run configuration:



The Status Bar

This area displays the state of the current execution status of projects being solved. It also shows the status of documents being edited in the Editing Area, and the current line and column number of the cursor.



Writable | Insert | 14 : 19 | OPL with scalable data: (100%)

Status Bar (partial view)

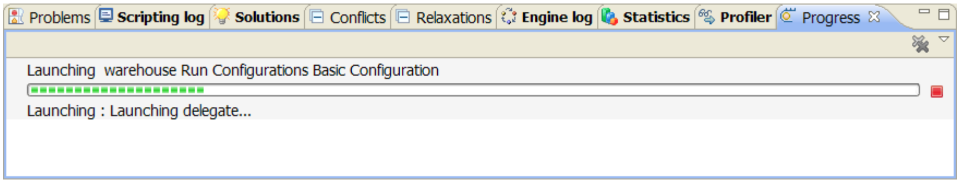
The middle part of the Status Bar (seen on the left in the illustration above), indicates that the file currently being edited is **Writable** (as opposed to **Read-Only**) and that the editor is in **Insert** mode (as opposed to **Overwrite** mode). The numbers indicate the line number and column number of the current cursor location.

The box at the right of the Status Bar displays an animated graphic while a project is running, and a static graphic when it is not, and a message is displayed beside the graphic indicating progress.

Run progress messages in the Status Bar

Whether the model is running in standard run mode or in debug, browse, or background mode, you can see messages about the progress of the run in the OPL IDE Status Bar:

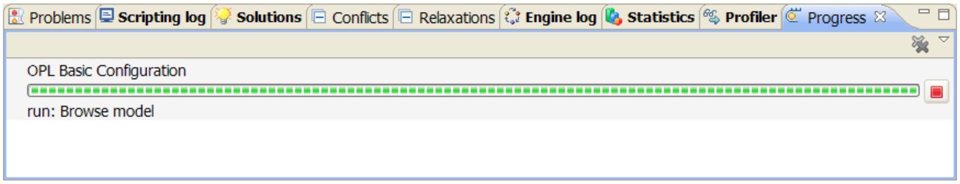
- When the solve begins, a **Launching <run configuration name>** message appears at the right of the Status Bar.
- As the solve progresses, the message changes to **<run configuration name> <percent>**, with a percentage displayed to indicate progress.
- When the solve is finished, the message changes to **<run configuration name> 100%**, to indicate that the run has completed
- In addition, as the model is solving, the run indicator becomes animated.
- If you click the **Show background operations** icon at the extreme right of the toolbar, a **Progress** tab appears in the Output Area. If the solve is still running, you see a display similar to the following:



Launching warehouse Run Configurations Basic Configuration

Launching : Launching delegate...

When the solve has finished, you see a display similar to this:

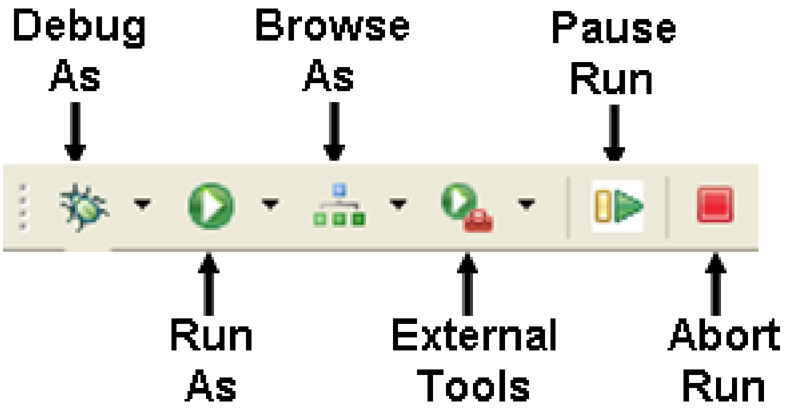


OPL Basic Configuration

run: Browse model

The Run execution toolbar options

In addition to the Run context menu in the OPL Projects Navigator, there are buttons in the execution toolbar that can be used to run your projects in different ways:



The execution toolbar

The four options available to you on the OPL execution toolbar give you a number of different ways to launch your projects:

- **Debug** — launches the project in debug mode.
- **Run** — launches the project in normal run mode.
- **Browse** — displays the project in the Problem browser, but without solving it.
- **External Tools** — launches the project in “background mode” in its own window in the Output Area. This is similar to launching a project using the `oplrun` command, allows you to continue working in the OPL IDE on the same project or a different project as the first project solves.

Runs can also be paused and resumed by clicking the **Pause** button, or stopped by using the **Abort** button ■ at the right of the execution toolbar. These buttons become active when a solve is in progress.

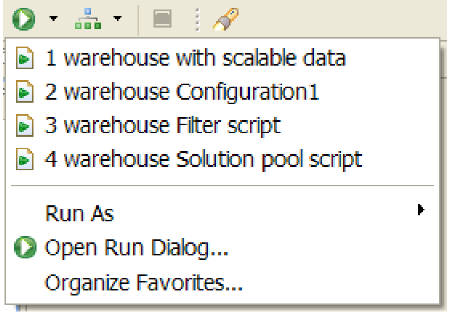
The first four buttons all work similarly. In this *Introduction to the OPL IDE*, we will describe only one of them, the **Run** button. Additional details can be found on each of the options in the *Execution toolbar* section of the *IDE Reference*.

The default behavior of the **Run** button is configurable. See *Run, Debug, Browse, and External Tools button options*for more information.

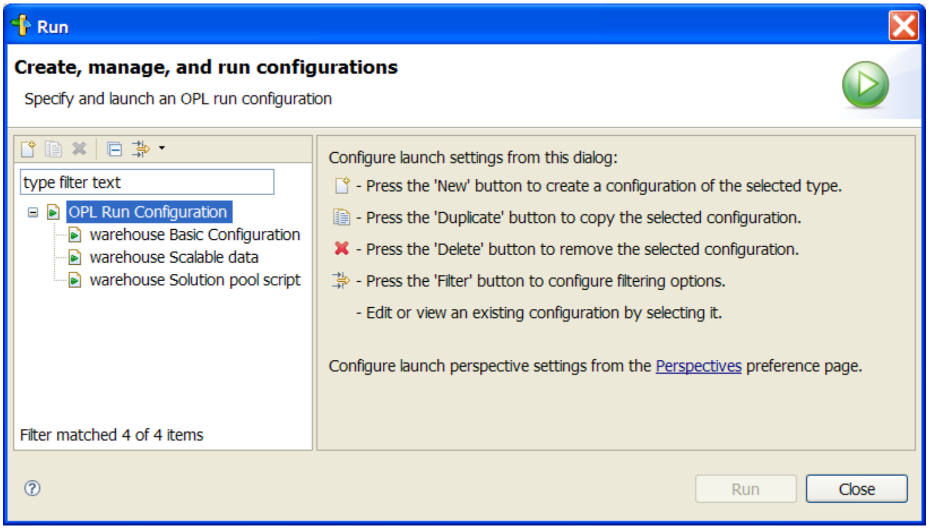
Working with the Run toolbar button:

♦ When you click the **Run** button itself, you launch the last run configuration executed.

♦ When you click the arrow to the right of the **Run** button, it expands to show you a list of your most recently accessed run configurations and other options:


1 warehouse with scalable data
2 warehouse Configuration1
3 warehouse Filter script
4 warehouse Solution pool script
Run As
Open Run Dialog...
Organize Favorites...

- **Recently run list** — This area at the top of the menu is populated with the names of the run configurations you have run in your current OPL session.
- **Run As** — This option works similarly to the **Run** command on the right-click context menu in the OPL Projects Navigator, and allows you to run favority run configurations that you have added to a list using the **Open Run Dialog** option.
- **Open Run Dialog** — This option launches a dialog window that allows you to create, manage and run different configurations.



Create, manage, and run configurations

Specify and launch an OPL run configuration

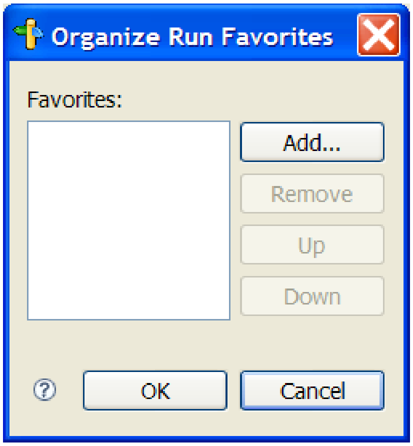
Filter matched 4 of 4 items

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

- **Organize Favorites** — This option launches a dialog window that allows you to organize your different run configurations.



Getting Started tutorial

A tutorial in which you launch the IDE, create an empty project, enter a model, add data, add a settings file, create run configurations and execute them.

In this section

Introduction

This *Getting Started* guide walks you through most features of the OPL IDE by means of a tutorial in which you create and execute a project. More tutorials elaborate on IDE features in *IDE Tutorials*.

Prerequisites - before you start

At this stage, it is assumed that you have already successfully installed ILOG OPL Development Studio on your platform.

Before starting to use the product, make sure you have installed the correct license key for your needs. For more information, see *Licensing Scheme*.

Read the *Quick Start* document for an introduction to how OPL handles optimization problems.

In addition, you should have read the *Introduction to the OPL IDE* section of this manual or the *OPL 6.x Migration Guide* or the copy of it in this manual, which contains important information about the OPL graphical user interface.

Make sure you read How to read the OPL documentation for details of prerequisites, conventions, documentation formats, and other general information.

At this point you are ready to launch the ILOG OPL IDE (or the IDE for short) as explained in the *Launching the OPL IDE* section of *Introduction to the OPL IDE*.

Once the IDE is open, you can either read *Tour of the OPL Graphical User Interface* in the *IDE Reference* to discover the graphical user interface or move on directly to *Creating a project* if you feel familiar enough with the interface and want to start working on a project immediately.

Creating a project

Walks you through creating a project file, adding model and data, and setting mathematical programming options.

In this section

Purpose

After launching the ILOG OPL IDE (or the IDE for short) as described in the *Launching the OPL IDE* section, you will want to solve a problem of your own. For this, you will first have to define a working document in the IDE. You can do this either by editing an existing project (see *Reusing existing files and projects* in the *IDE Reference*) or by starting your own project.

In this tutorial, you will start your own project. To do this, you will:

- Create a project: see *Creating an empty project*.
- Add an existing model or write a new one: see *The pasta production example*.
- Add one or more existing data files or write new ones: see *Adding data*.
- Set mathematical programming options, if applicable: see *Changing an MP option value*.

Once your project is built, you will populate a run configuration, execute it, and study the results as explained in *Executing a project* and *Examining a solution to the model*.

The pasta production example

You could write your own model from scratch by following the syntax rules from the *Language Reference Manual* and the *Language User's Manual*, but since this tutorial does not aim at teaching you the modeling or scripting languages, you are going to reuse the pasta production example, described in *A production problem* in the *Language User's Manual*, for the purpose of this exercise.

> **Note**: The pasta production model is designed to be solved by the CPLEX engine. However, the content of this section would be the same for a model solved by the CP Optimizer engine, except where explicitly mentioned.

The problem is 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.

The code extract below (`product.mod` file) shows an OPL model (the tuple version) for this example. This model is part of the `production` project, which is available at the following location:

`<OPL_dir>\examples\opl\production`

where `<OPL_dir>` is your installation directory.

OPL model for the production planning example (`product.mod`)

```
{string} Products = ...;
{string} Resources = ...;
tuple productData {
   float demand;
   float insideCost;
   float outsideCost;
   float consumption[Resources];
}
productData Product[Products] = ...;
float Capacity[Resources] = ...;

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

execute CPX_PARAM {
  cplex.preind = 0;
  cplex.simdisplay = 2;
}


minimize
  sum( p in Products )
    (Product[p].insideCost * Inside[p] +
    Product[p].outsideCost * Outside[p] );
subject to {
  forall( r in Resources )
    ctInside:
      sum( p in Products )
        Product[p].consumption[r] * Inside[p] <= Capacity[r];
  forall( p in Products )
    ctDemand:
      Inside[p] + Outside[p] >= Product[p].demand;
}
```

The following code extract (`product.dat` file) shows the data initialization for the problem.

OPL data for the production planning example (`product.dat`)

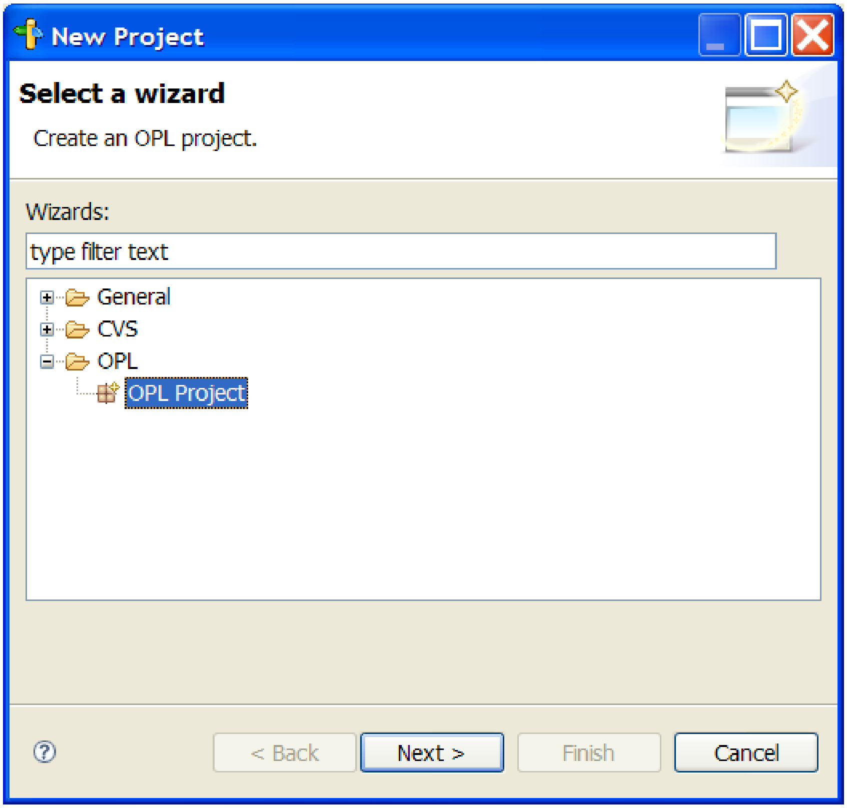
```
Products =  { "kluski", "capellini", "fettucine" };
Resources = { "flour", "eggs" };
Product = #[
      kluski : < 100, 0.6, 0.8, [ 0.5, 0.2 ] >,
      capellini : < 200, 0.8, 0.9, [ 0.4, 0.4 ] >,
      fettucine : < 300, 0.3, 0.4, [ 0.3, 0.6 ] >
          ]#;
Capacity = [ 20, 40 ];
```

Creating an empty project

To start from an empty project:

1. In the main menu, choose **File>New>OPL Project**.

   The first screen of the New Project wizard is displayed:



New Project

Select a wizard

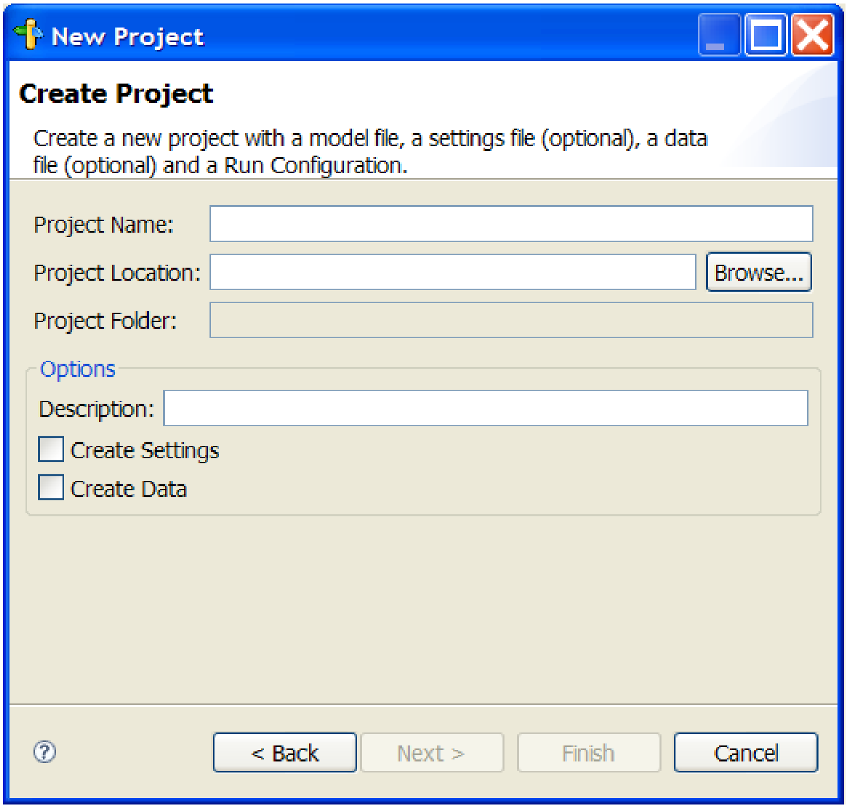
Create an OPL project.

Wizards:

type filter text

2. Open the **OPL** entry and select **OPL Project**. Then click **Next**.

   The second screen of the New Project wizard is displayed.



New Project

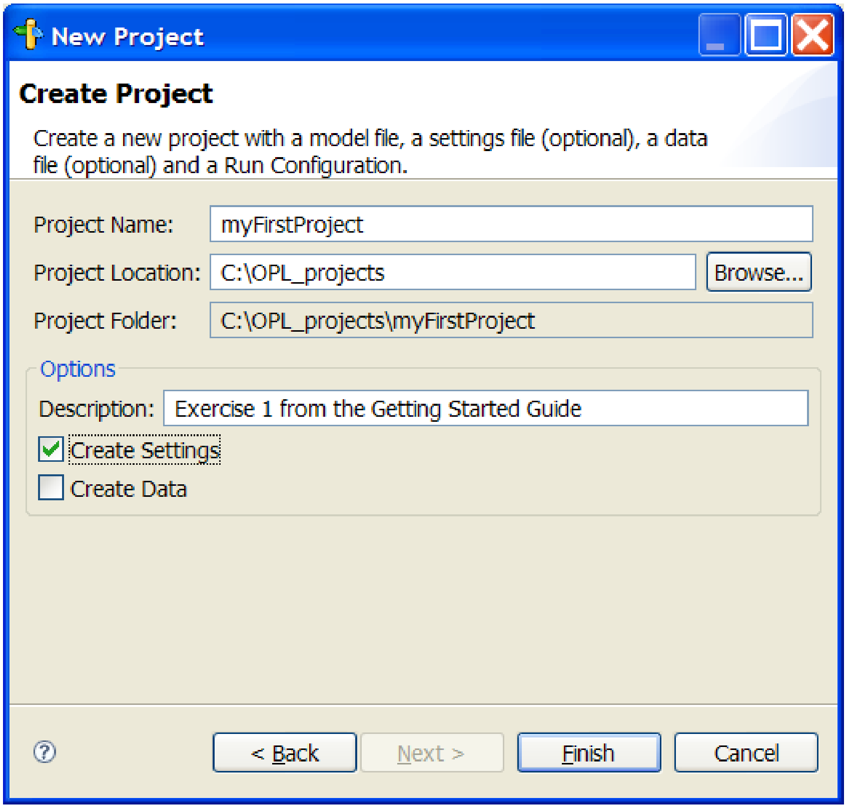
Create Project

Create a new project with a model file, a settings file (optional), a data file (optional) and a Run Configuration.

3. Enter the following information into the New Project window:

- Type **myFirstProject** as the **Project Name** of your new project.
- Enter a destination **Project Location** for the project, *other* than the OPL examples directory. For example, create a directory named `C:\OPL_projects` and enter it in this field.
- Enter a **Description** for the project.
- Check the **Create settings** option box. This enables you to automatically create and associate a *settings file* with your project when you create it.
- Check the **Create Settings** option, to create a blank settings file for the project.
- Leave the **Create Data** option unchecked, because you will later be adding existing data files to the project rather than typing them in. This option, if checked, enables you to automatically create a blank *data file* for your project.

Your window should look similar to the one shown below.



New Project

Create Project

Create a new project with a model file, a settings file (optional), a data file (optional) and a Run Configuration.

Project Name: myFirstProject

Project Location: C:\OPL_projects

Project Folder: C:\OPL_projects\myFirstProject

Options

Description: Exercise 1 from the Getting Started Guide

☑ Create Settings

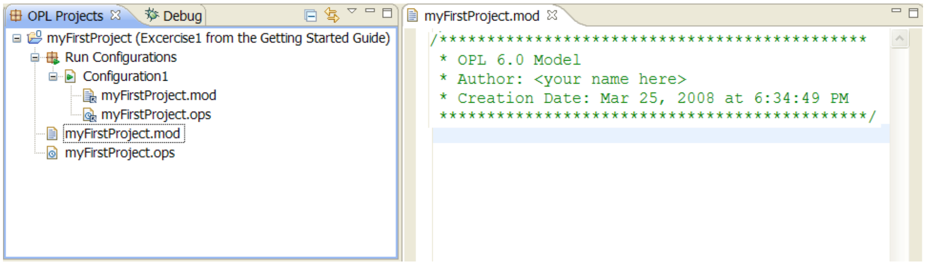
□ Create Data

Creating a project

4. When you have entered all the information, click **Finish**.

Note: If a project with the same name already exists in OPL, a message warns you and the **Finish** button remains greyed out as long as you don't enter a unique name.

The project is created, containing the model and settings files you specified, and appears in the OPL Projects Navigator. The new empty model file is displayed in the Editing Area:



```
/*********************************************
 * OPL 6.0 Model
 * Author: <your name here>
 * Creation Date: Mar 25, 2008 at 6:34:49 PM
 *********************************************/
```

New project and new empty model in main window

Note that the `.mod` and `.ops` extensions are automatically appended to the file names in the OPL Projects Navigator and in Windows Explorer (see *File types* in the *IDE Reference* and *Understanding OPL projects* in *Quick Start* for more information on file name extensions in OPL).

The OPL Projects Navigator displays a minimal tree containing:

- an empty `.mod` file: you are going to fill it with OPL statements in the next step, *Adding the model*
- an `.ops` file that reflects all the default values of the dialog box for MP options, CP settings, and OPL settings: see *Changing an MP option value* in this manual and *Setting programming options* in the *IDE Reference*.
- the **Run Configurations** folder
- one run configuration containing the model and settings files you have just created (see *New project and new empty model in main window*).

Important: The only mandatory component in a project or run configuration is a valid model file. A project can contain more than one model, but a run configuration can contain only one.

See also *The main window* in the *IDE Reference* for reference information on the graphical user interface.

Adding the model

You are going to copy-paste the pasta production model into the Editing Area.

To add the pasta production model to the project:

1. Choose **File>Open File in Editor**, and browse to

   ```
   <OPL_dir>\examples\opl\production\product.mod
   ```

   then double-click the filename in the dialog box or select it and click **Open**.

   > **Note**: There are two model files in this project. Make sure that you open `product.mod` and not `production.mod`.

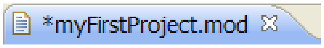
   The OPL statements of the `product.mod` file appear in the Editing Area, in a separate window. If you click the tab of the `myFirstProject.mod` file, you can see that your empty model is in a different editor.

2. Click in the `product.mod` window and press **Ctrl-A**, then **Ctrl-C** to copy the contents of the file.

3. Click the tab of the `myFirstProject.mod` file to redisplay the empty editing window for your own project.

4. Place your cursor after the header comments and press **Ctrl-V** to paste the copied statements into the `myfirstproject.mod` file.

   Note that an asterisk (*****) appears in the tab of this window.

   This indicates that the file is unsaved. Click the **Save** button 💾 to save the file.

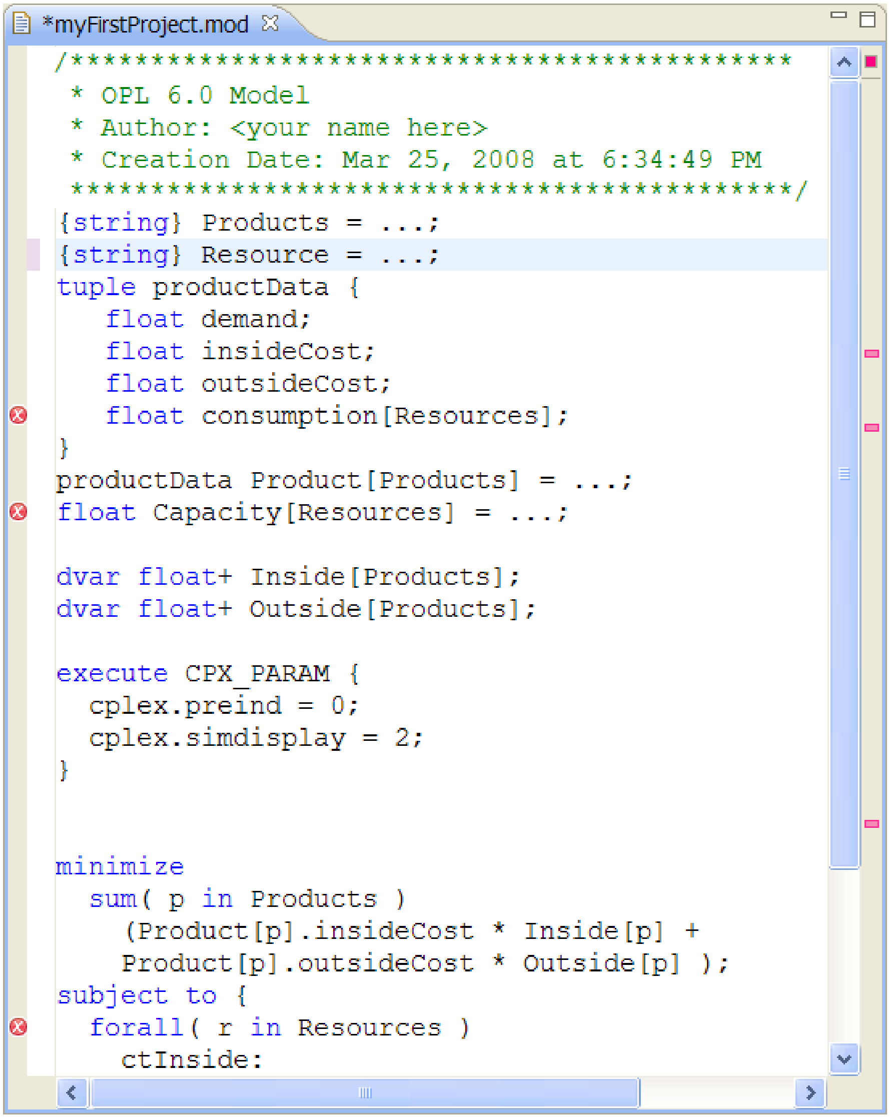
5. Close the open window for the `product.mod` file in the Editing Area by clicking the **X** button ☒ to the right of its tab. Leave the edit window for the `myFirstProject.mod` file open for the next exercise.

Dealing with errors

For this very first start with ILOG OPL, you have copied and pasted an existing model for quicker results. In your real business life, however, you will enter OPL statements from the keyboard. By default, the IDE checks for syntax and semantic errors automatically as you type and error messages are displayed in the **Problems** tab at the bottom of the IDE.

To observe the default behavior:

1. In the second line of `product.mod` displayed in the Editing Area, remove the “s” from the middle of the word “Resources”.

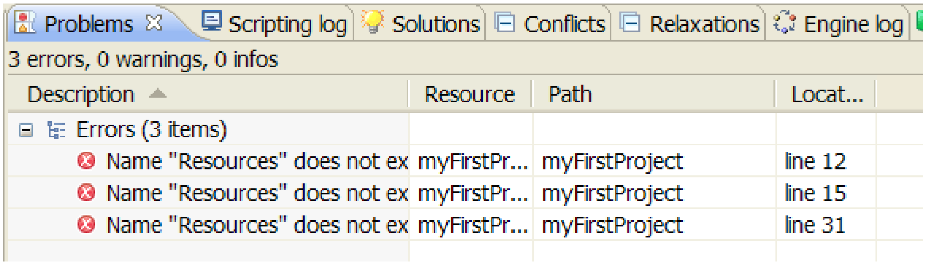


*myFirstProject.mod

```
/**********************************************
 * OPL 6.0 Model
 * Author: <your name here>
 * Creation Date: Mar 25, 2008 at 6:34:49 PM
 *********************************************/
{string} Products = ...;
{string} Resource = ...;
tuple productData {
   float demand;
   float insideCost;
   float outsideCost;
   float consumption[Resources];
}
productData Product[Products] = ...;
float Capacity[Resources] = ...;

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

execute CPX_PARAM {
  cplex.preind = 0;
  cplex.simdisplay = 2;
}


minimize
  sum( p in Products )
    (Product[p].insideCost * Inside[p] +
    Product[p].outsideCost * Outside[p] );
subject to {
  forall( r in Resources )
    ctInside:
```

A syntax error

The line containing the error is highlighted and for this line any other line affected by the error, a red error symbol appears in the margin ⊗ . The **Problems** tab immediately displays the corresponding error messages, indicating the description, location, and source.



3 errors, 0 warnings, 0 infos

Description	Resource	Path	Locat...
Errors (3 items)			
Name "Resources" does not ex	myFirstPr...	myFirstProject	line 12
Name "Resources" does not ex	myFirstPr...	myFirstProject	line 15
Name "Resources" does not ex	myFirstPr...	myFirstProject	line 31

Problems tab

2. Remove the mistake. The error message disappears and the indicators in the editor disappear.

3. Press **Ctrl+S** or choose **File>Save** or press the **Save** button in the standard toolbar to save your work.

In the next step, you will add two data files to the project.

Adding data

You can add more than one data file to a project or run configuration. If you do so, their order is meaningful. See *Ordering files within a run configuration* in the *Introduction to the OPL IDE* section for more information. You can either add existing data files or create them as you add them. In this tutorial, you are going to add two existing data files to your project.

> **Note**:
> 1. Until you add it to a project, a data file does not appear in the project tree.
> 2. All files pertaining to the same project must be stored within the same parent project directory.

This part of the tutorial assumes you have created a project and is meaningful only if the model is not empty.

To add a data file to the project:

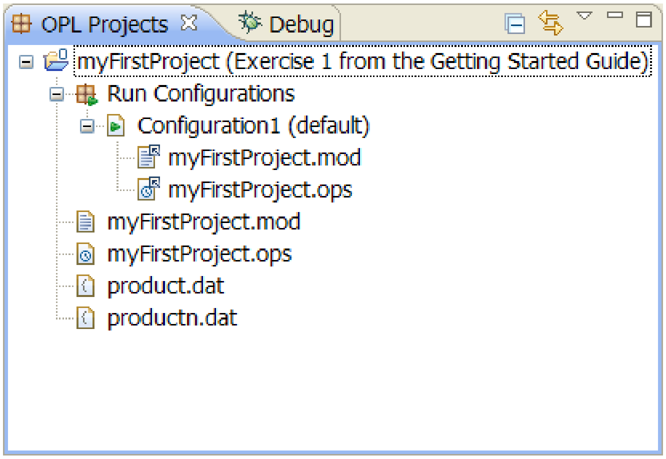
1. Open a Windows Explorer window and navigate to the `<OPL_dir>\examples\opl\production` directory.
2. *Drag* the `product.dat` file from Windows Explorer into the OPL Projects Navigator, and drop it on the **myFirstProject** project folder.

   The data file is added to the project but is not automatically added to the run configuration.

   > **Note**: You could also have used the **File>Copy Files to Project** menu command to add these files to your project.

3. Follow the same steps to add the `productn.dat` file from the same location.

   Your OPL Projects Navigator should now look like this:



myFirstProject (Exercise 1 from the Getting Started Guide)
Run Configurations
Configuration1 (default)
myFirstProject.mod
myFirstProject.ops
myFirstProject.mod
myFirstProject.ops
product.dat
productn.dat

Adding data files to a project

You are now going to execute your project. Later, you will modify the settings file so that you can use it to execute different run configurations of your project (see *Creating and executing a different configuration*).

Executing a project

Walks you through populating and executing the run configuration, creating a different configuration, and understanding project execution.

In this section

What you are going to do

The OPL IDE has a **Run** button which starts the solving engine to find a solution to the problem expressed in the active model. Solving a model in ILOG OPL Development Studio consists of executing the corresponding project, more precisely a run configuration of it; that is, a subset of the model, data, and settings files that make up your project.

Clicking the **Run** button executes the last run configuration launched.

If you want to execute any run configuration other than the last run configuration you launched, it is probably better to use the **Run** option of the right-click context menus in the OPL Projects Navigator to launch the exact run configuration you want. See *The Run options* section for more details.

Note: To execute a run configuration in debugging mode with breakpoints, you would use the **Debug** button instead of the **Run** button. See *Using ILOG Script for OPL* in *IDE Tutorials*.

To continue with the production planning tutorial, you are now going to:

♦ Populate the run configuration of your project: see *Populating and executing the run configuration*

♦ Create more configurations to execute your model with different data and/or settings: see *Creating and executing a different configuration*

♦ Learn more about model solving: see *What happens when you execute a run configuration* in the *IDE Reference*.

Populating and executing the run configuration

When you are finished creating an OPL project, the OPL Projects Navigator should typically look as shown in the figure *Adding data files to a project*. You have defined the project as a set of model, data, and settings but the run configuration contains only the `.mod` and `.ops` files. You need to add the data you want to try with your model. Populating a run configuration consists therefore in adding data and/or settings files to the run configuration subtree.

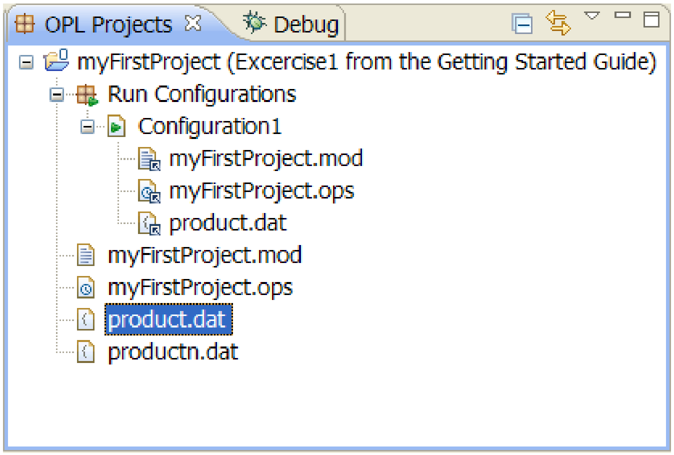
A run configuration must contain at least a model and can contain only one model.

To populate a run configuration:

1. In the OPL Projects Navigator, drag and drop the `product.dat` file to the **Configuration1** run configuration.

> **Note**: If you inadvertently drop the wrong file or drop a file to the wrong place, you can at any time right-click it and choose **Delete**, then confirm. This does not remove the file from the disk.

2. The OPL Projects Navigator now displays the data file name in the run configuration:



```
myFirstProject (Excercise1 from the Getting Started Guide)
    Run Configurations
        Configuration1
            myFirstProject.mod
            myFirstProject.ops
            product.dat
    myFirstProject.mod
    myFirstProject.ops
    product.dat
    productn.dat
```

Adding a data file to a run configuration

3. Right-click **Run Configurations** and select **Run>Configuration1**.

Note: A project can contains more than one run configuration. To make a configuration the default, right-click its name and choose **Set as Default**. But you can execute any of the other run configurations using the **Run** option on this menu.

You can also execute any run configuration, whether default or not, by clicking the arrow next to the **Run** button ▶ ▾ and selecting its name from the **Run** option on that contextual menu.

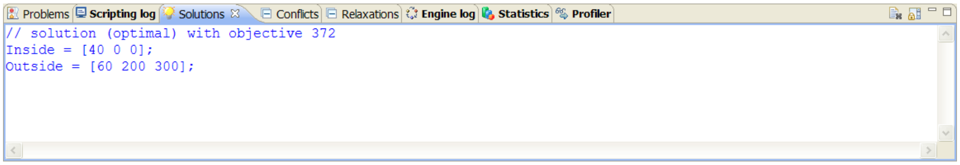
See *The Run options* for more information on running OPL models.

See *The Status Bar* for more information on obtaining run status information during the run.

4. Observe the Output Area.

The highlighted tab names show which output panels have received content during execution. The pasta production model uses CPLEX as the solving engine.

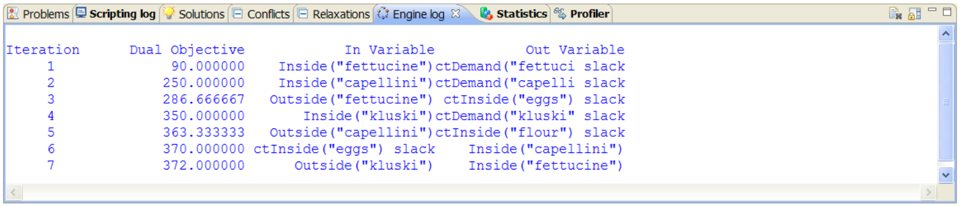
♦ The **Solutions** tab displays one solution.



```
// solution (optimal) with objective 372
Inside = [40 0 0];
Outside = [60 200 300];
```

Solution for Configuration1

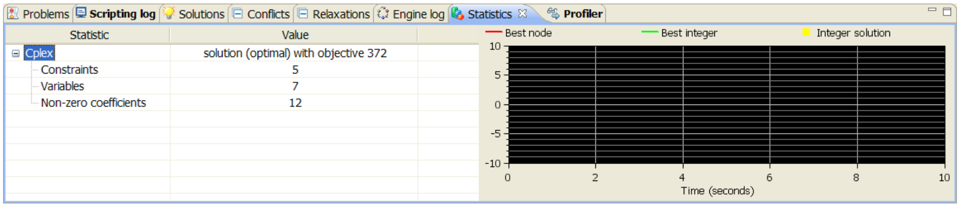
♦ The **Engine Log** tab displays details for each iteration.



```
Iteration      Dual Objective            In Variable           Out Variable
     1              90.000000    Inside("fettucine")ctDemand("fettuci slack
     2             250.000000    Inside("capellini")ctDemand("capelli slack
     3             286.666667   Outside("fettucine") ctInside("eggs") slack
     4             350.000000       Inside("kluski")ctDemand("kluski" slack
     5             363.333333   Outside("capellini")ctInside("flour") slack
     6             370.000000 ctInside("eggs") slack    Inside("capellini")
     7             372.000000      Outside("kluski")    Inside("fettucine")
```

Engine Log for BasicConfiguration (MP)

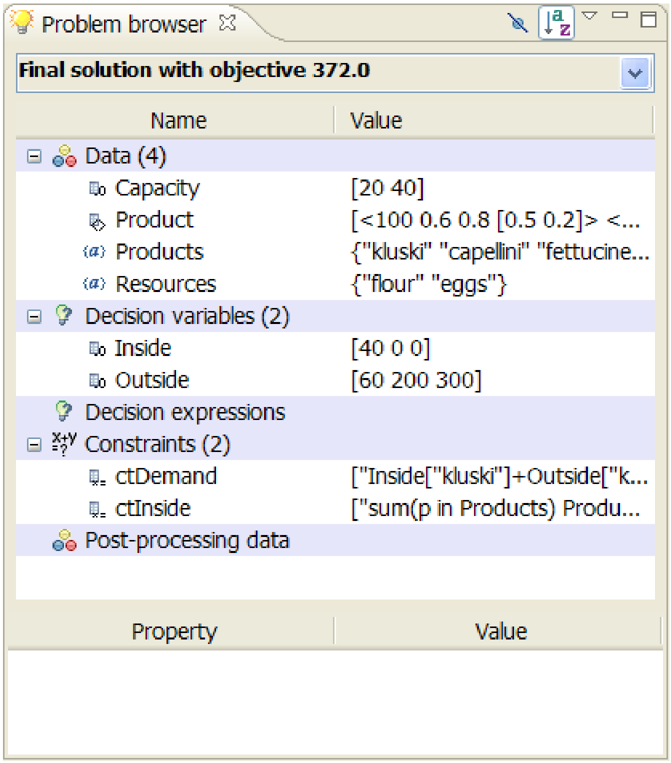
♦ The **Statistics** tab shows, among other information, the algorithm and the number of iterations.



Statistic	Value
Cplex	solution (optimal) with objective 372
Constraints	5
Variables	7
Non-zero coefficients	12

Statistics for BasicConfiguration (MP)

- ♦ Also, notice that the Problem Browser now contains data for the problem, including the solution, displayed in the drop-down at the top. You will learn more about the Problem Browser in later sections of this tutorial and in the *IDE Tutorials*.



Final solution with objective 372.0

Name	Value
Data (4)	
Capacity	[20 40]
Product	[<100 0.6 0.8 [0.5 0.2]> <...
Products	{"kluski" "capellini" "fettucine...
Resources	{"flour" "eggs"}
Decision variables (2)	
Inside	[40 0 0]
Outside	[60 200 300]
Decision expressions	
Constraints (2)	
ctDemand	["Inside["kluski"]+Outside["k...
ctInside	["sum(p in Products) Produ...
Post-processing data	

Property	Value

See *The Output Area* in the *IDE Reference* and *Understanding the Statistics and Progress Chart (MP models)* in *IDE Tutorials* for more information.

You can now continue with the tutorial and create another run configuration to learn more about project settings, or you may want to go first to *What happens when you execute a run configuration* in the *IDE Reference* to learn more on the execution process, then proceed to *Examining a solution to the model* to understand results.

Adding a settings file

A settings file is where you store user-defined values of OPL options for language output, mathematical programming, or constraint programming. For more information, see*Preferences and options* in the *IDE Reference*.

> **Note**: If your model contains a `main` flow control script, the OPL values you set in the `.ops` file, as well as the settings set within the `main` script, apply to the current model only, not to the submodels loaded and solved at execution time.

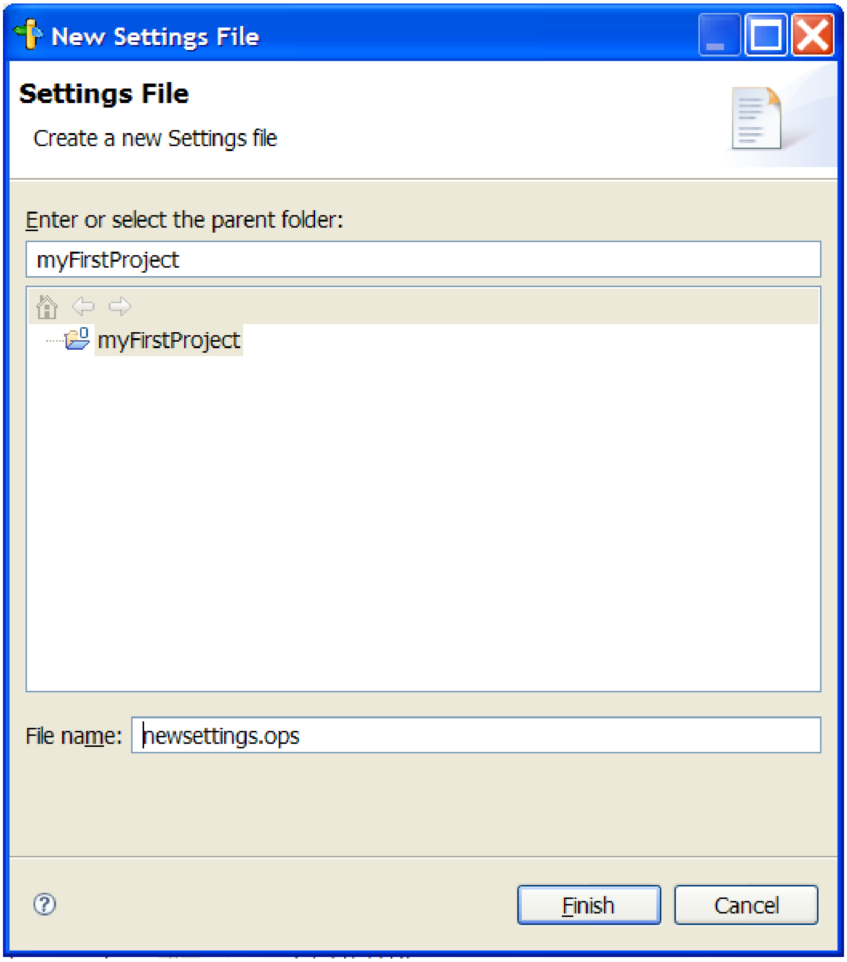
Because you left the **Create settings** option checked in Step 2 of *Creating an empty project*, a default settings file already exists for your project and this is the one you used in *Populating and executing the run configuration*.

To practice with a different run configuration without losing your default settings, you are now going to add a second settings file to your project and use it to set a different value to one mathematical programming option.

This stage of the tutorial assumes you have at least a model in your project and want to be able to modify OPL, CPLEX, or CP Optimizer parameters.

To add a settings file to an existing project:

1. Select the project name in the OPL Projects Navigator, then right-click and select **New>Settings**.
2. In the dialog box, select the parent project and provide a name for the new settings file. Then click **Finish**. The `.ops` extension will be added automatically.



3. Notice the changes in IDE window:

   - The `.ops` file is added to the project in the OPL Projects Navigator.
   - The settings editor appears.
   - The Outline window displays the settings outline. You can later access the modified settings directly from this window.

You are now ready to use the new settings file to set a mathematical programming option with which you will then execute the model.

Changing an MP option value

You are now going to modify the default value of one of the OPL MP options.

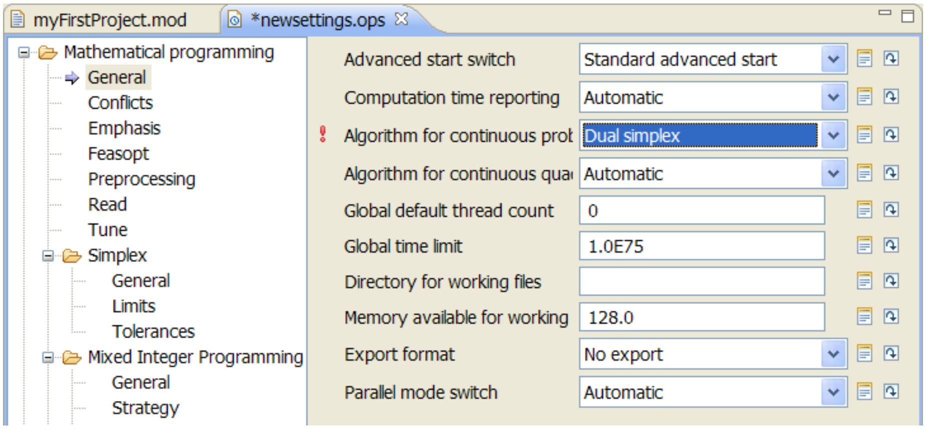
To change an option value in the IDE:

1. Double-click the new `.ops` file in the OPL Projects Navigator, if it is not already open.

   The panel for mathematical programming options is displayed in the Editing Area.

2. In the **Mathematical Programming /General** category, choose **Dual simplex** from the **Algorithm for continuous problems** list.

   Notice the red exclamation mark indicating that a default value has been changed.

Advanced start switch	Standard advanced start
Computation time reporting	Automatic
! Algorithm for continuous prot	Dual simplex
Algorithm for continuous qua	Automatic
Global default thread count	0
Global time limit	1.0E75
Directory for working files	
Memory available for working	128.0
Export format	No export
Parallel mode switch	Automatic

   Each option available is documented individually in OPL language options, Constraint programming options, and Mathematical programming options, in the *IDE Reference*.

3. Choose **File>Save**.

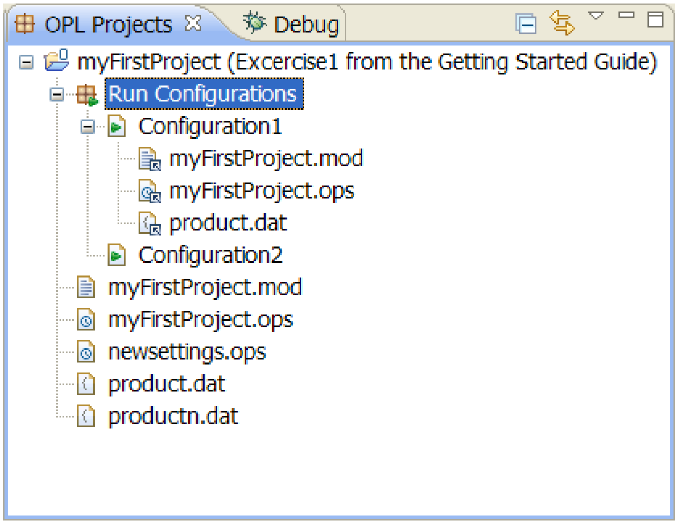
Your project now includes a model file, two data files, and two settings files. In the next step, *Creating and executing a different configuration*, you will create a second run configuration to execute the model with different data and different settings.

Creating and executing a different configuration

You will now create a second run configuration, then populate and execute it as you did in *Populating and executing the run configuration*.

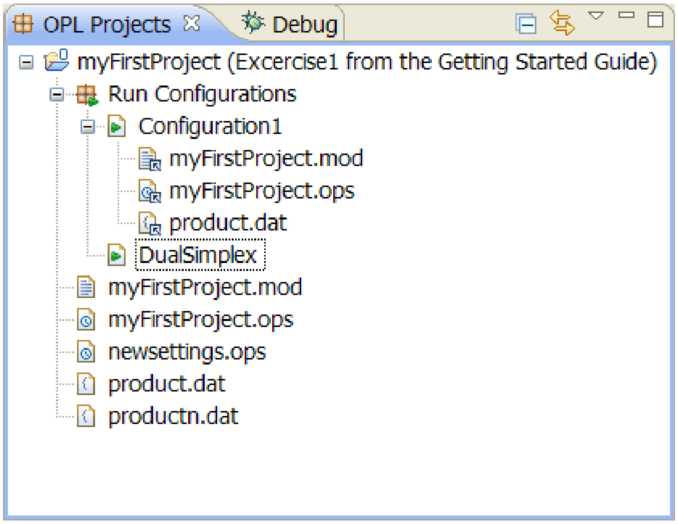
To create and execute a second run configuration:

1. In the selected project, right-click the **Run Configurations** folder and choose **New Run Configuration** from the context menu. A new run configuration with the default name **Configuration2** is added.



- myFirstProject (Excercise1 from the Getting Started Guide)
  - Run Configurations
    - Configuration1
      - myFirstProject.mod
      - myFirstProject.ops
      - product.dat
    - Configuration2
  - myFirstProject.mod
  - myFirstProject.ops
  - newsettings.ops
  - product.dat
  - productn.dat

2. Optionally, rename the run configuration, by selecting it and right-clicking and choosing **Properties** from the context menu. In this example, **Configuration2** is renamed **DualSimplex**.

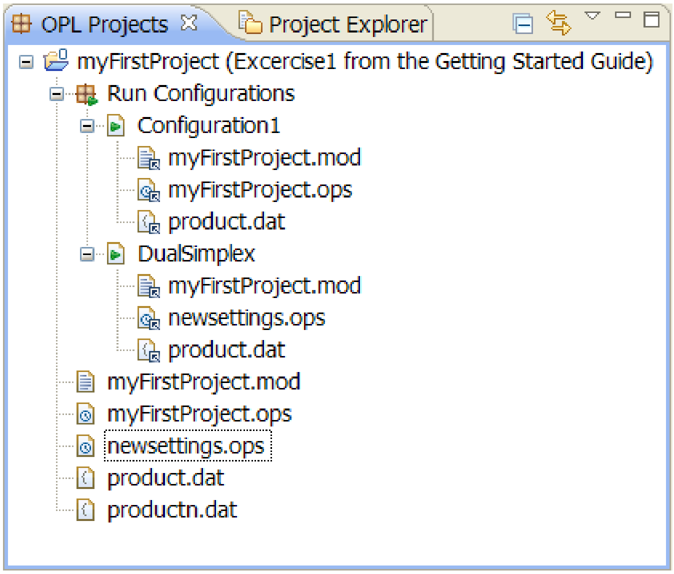


OPL Projects ☒ | Debug

- myFirstProject (Excercise1 from the Getting Started Guide)
  - Run Configurations
    - Configuration1
      - myFirstProject.mod
      - myFirstProject.ops
      - product.dat
    - DualSimplex
  - myFirstProject.mod
  - myFirstProject.ops
  - newsettings.ops
  - product.dat
  - productn.dat

3. Drag and drop the files `myFirstProject.mod`, `product.dat`, and your new settings file`newsettings.ops` into the new **DualSimplex** run configuration.

The OPL Projects Navigator should look like this.



OPL Projects ☒ | Project Explorer

- myFirstProject (Excercise1 from the Getting Started Guide)
  - Run Configurations
    - Configuration1
      - myFirstProject.mod
      - myFirstProject.ops
      - product.dat
    - DualSimplex
      - myFirstProject.mod
      - newsettings.ops
      - product.dat
  - myFirstProject.mod
  - myFirstProject.ops
  - newsettings.ops
  - product.dat
  - productn.dat

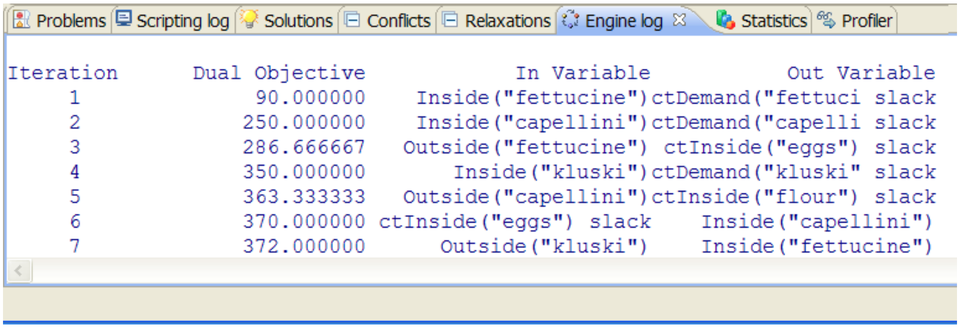
A run configuration with different settings

4. Right-click in the project and select **Run>DualSimplex** from the context menu to execute the model with changed settings.

The CPLEX engine now uses **Dual simplex** as the value of the algorithm for continuous problems.

Observe the differences in the output tabs.

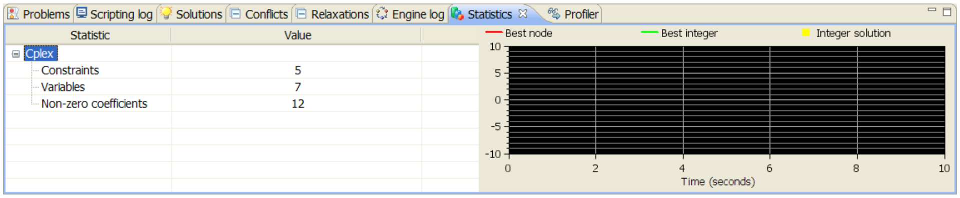
- The **Solutions** tab displays the same solution (see *Solution for Configuration1*).
- The **Engine Log** tab shows a different report.



```
Iteration      Dual Objective            In Variable           Out Variable
     1           90.000000   Inside("fettucine")ctDemand("fettuci slack
     2          250.000000   Inside("capellini")ctDemand("capelli slack
     3          286.666667  Outside("fettucine") ctInside("eggs")  slack
     4          350.000000      Inside("kluski")ctDemand("kluski"  slack
     5          363.333333  Outside("capellini")ctInside("flour")  slack
     6          370.000000 ctInside("eggs") slack    Inside("capellini")
     7          372.000000      Outside("kluski")    Inside("fettucine")
```

Engine Log for DualSimplex configuration

- The **Statistics** tab shows a different algorithm (Dual Phase2) and number of iterations.



Statistic	Value
⊟ Cplex	
Constraints	5
Variables	7
Non-zero coefficients	12

Statistics for DualSimplex configuration

Examining the Statistics and Progress Chart (MP) in *IDE Tutorials*.

You can proceed to *Examining a solution to the model* to learn more about the execution process and its results. See *What happens when you execute a run configuration* in the *IDE Reference* for details on the execution process.

Examining a solution to the model

Explains how to read the solutions in the output tabs and read details of the executed model in the Problem Browser.

In this section

Execution results

When you execute a run configuration to find the solutions to the problem expressed by the model, the IDE provides facilities for examining results — one for examining the solution and one for examining the details of your model.

- **To examine the solution**, browse the output tabs in the lower half of the main window, as explained in the next section *The Output tabs*.

  You can customize the solution display: see OPL language options in the *IDE Reference*.

- **To examine the structure** of the model as solved by the engine, use the Problem Browser: see *Understanding the Problem Browser* and *Doing more with the Problem Browser* in the *IDE Reference*.

- **Tuning parameters:** When working on a MIP project, possibly by trying various run configurations, you may want to test the performance of your model before deployment. The **Tune model** button in the IDE standard toolbar offers a convenient way to do so. See *Using the performance tuning tool* in *IDE Tutorials*.

Note: For demonstration purposes, the illustrations used in this section are taken from various code samples from the product distribution. However, you can continue with the project you have just created.

The Output tabs

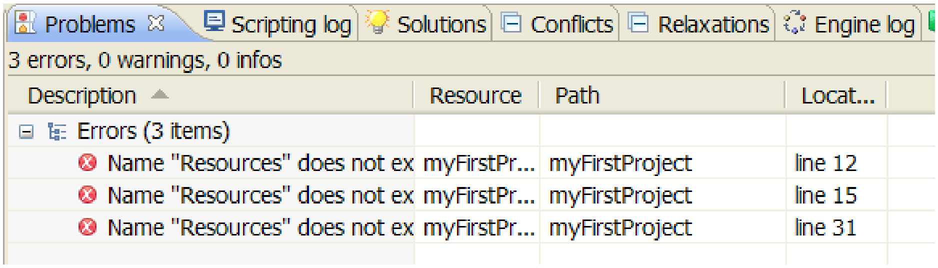
After you execute a run configuration, the IDE searches for the optimal solution, and when execution is complete, fills several output tabs in the main window.

These tabs are:

- *Problems*
- *Scripting log*
- *Solutions*
- *Conflicts*
- *Relaxations*
- *Engine Log*
- *Statistics*
- *Profiler*

Problems

The **Problems** tab displays semantic and syntax errors as you type when you write a model manually, and internal errors, such as scripting or algorithm errors, when you solve a model.



3 errors, 0 warnings, 0 infos

Description	Resource	Path	Locat...
Errors (3 items)			
Name "Resources" does not ex	myFirstPr...	myFirstProject	line 12
Name "Resources" does not ex	myFirstPr...	myFirstProject	line 15
Name "Resources" does not ex	myFirstPr...	myFirstProject	line 31

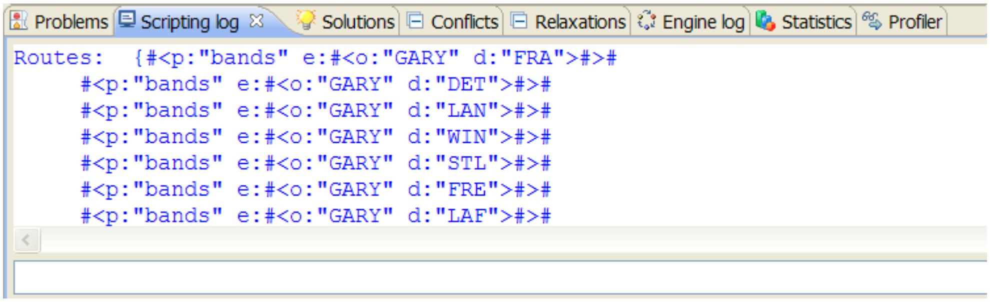
Problems tab

How to read messages:

- The **Description** column guides the user as to the nature of the error.
- The **Resource** column indicates which resource the error occurred in.
- The **Path** column provides the path to the current file.
- The **Location** column reflects the line that’s affected by the error.

Scripting log

The **Scripting log** tab shows execution output related to the ILOG Script `main` or `execute` or `prepare` blocks of the model (if applicable).

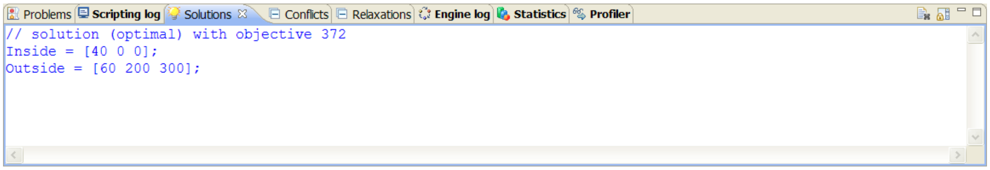


```
Routes:   {#<p:"bands" e:#<o:"GARY" d:"FRA">#>#
      #<p:"bands" e:#<o:"GARY" d:"DET">#>#
      #<p:"bands" e:#<o:"GARY" d:"LAN">#>#
      #<p:"bands" e:#<o:"GARY" d:"WIN">#>#
      #<p:"bands" e:#<o:"GARY" d:"STL">#>#
      #<p:"bands" e:#<o:"GARY" d:"FRE">#>#
      #<p:"bands" e:#<o:"GARY" d:"LAF">#>#
```

Scripting log tab (`transp4.mod`)

Solutions

The **Solutions** tab displays the final solution to a model and, if applicable, any intermediate feasible solutions found.



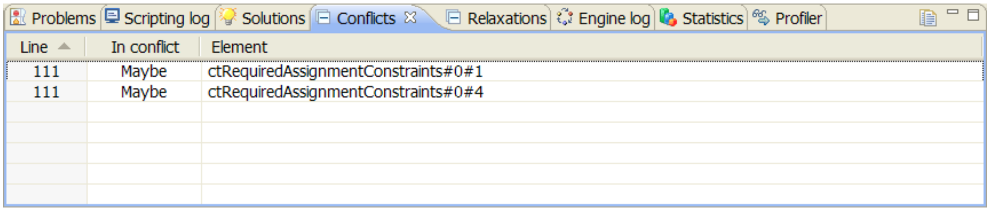
```
// solution (optimal) with objective 372
Inside = [40 0 0];
Outside = [60 200 300];
```

Solutions tab (basic configuration of `product.mod`)

By default the variables are displayed as well. You can disable this display, see OPL language options in the *IDE Reference*.

Conflicts

When a CPLEX model proves infeasible, the **Conflicts** tab shows the places where you can change the data or the way filtering constraints are expressed so as to remove the incompatibilities that made the model infeasible. See *Relaxing infeasible models* in *IDE Tutorials* for details.



Line	In conflict	Element
111	Maybe	ctRequiredAssignmentConstraints#0#1
111	Maybe	ctRequiredAssignmentConstraints#0#4

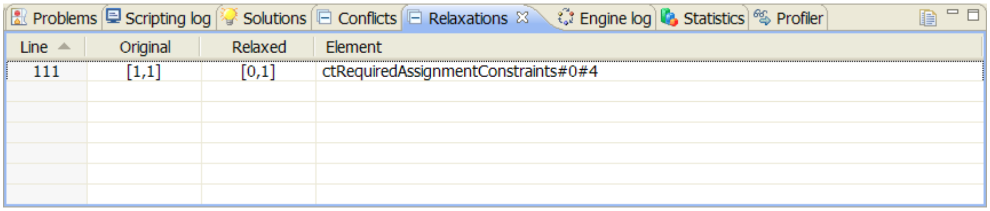
Conflicts tab (`nurses` project)

The **Conflicts** tab is empty after execution of `product.mod` because that project is not designed as infeasible.

Note: There is no support for conflicts for models solved by the CP Optimizer engine.

Relaxations

When a CPLEX model proves infeasible, the **Relaxations** tab shows the places that constraints can be relaxed to remove the incompatibilities that made the model infeasible. See *Relaxing infeasible models* in *IDE Tutorials* for details.



Line	Original	Relaxed	Element
111	[1,1]	[0,1]	ctRequiredAssignmentConstraints#0#4

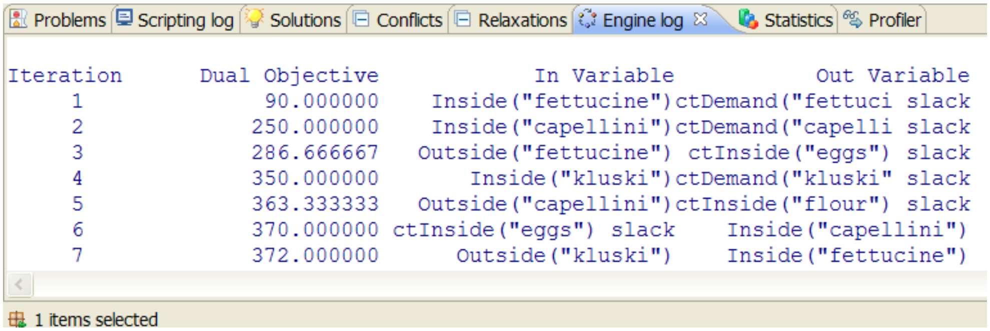
Relaxations tab (`nurses` project)

The **Relaxations** tab is empty after execution of `product.mod` because that project is not designed as infeasible.

Note: There is no support for relaxations for models solved by the CP Optimizer engine.

Engine Log

The **Engine Log** tab displays information from the solving engine (CPLEX for `product.mod`) on the solving process and on the objective function (in this example, a `minimize` statement).

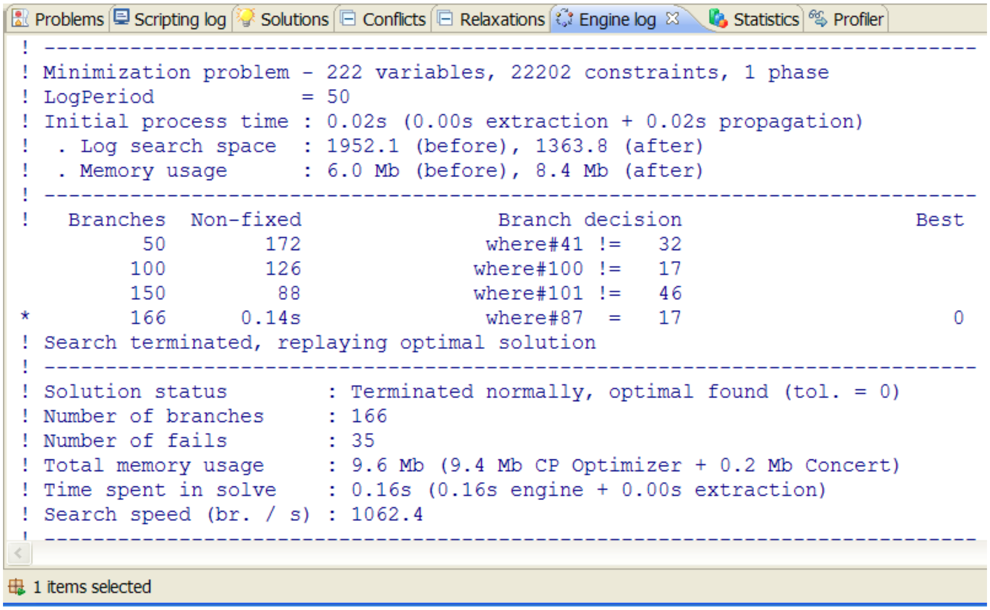


```
Iteration      Dual Objective            In Variable           Out Variable
    1            90.000000    Inside("fettucine")ctDemand("fettuci slack
    2           250.000000    Inside("capellini")ctDemand("capelli slack
    3           286.666667   Outside("fettucine") ctInside("eggs") slack
    4           350.000000       Inside("kluski")ctDemand("kluski" slack
    5           363.333333   Outside("capellini")ctInside("flour") slack
    6           370.000000 ctInside("eggs") slack     Inside("capellini")
    7           372.000000      Outside("kluski")     Inside("fettucine")
```

Engine Log for an MP model - CPLEX Dual Simplex (`product.mod`)

CPLEX users may recognize this information as what they see when executing CPLEX Interactive.

For comparison, the **Engine Log** for a constraint programming model looks like this:

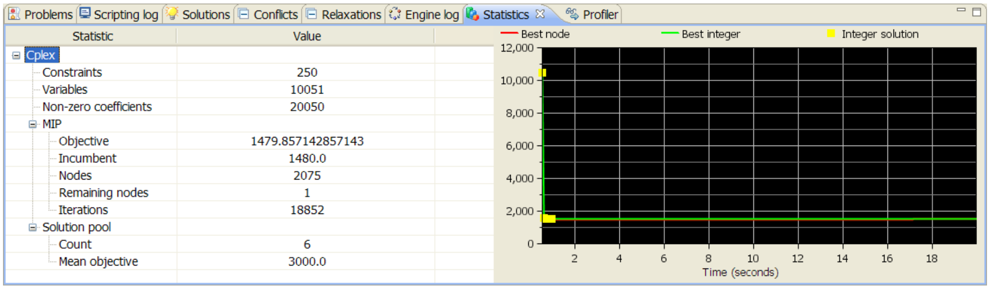


```
 ! ----------------------------------------------------------------------------
 ! Minimization problem - 222 variables, 22202 constraints, 1 phase
 ! LogPeriod            = 50
 ! Initial process time : 0.02s (0.00s extraction + 0.02s propagation)
 !  . Log search space  : 1952.1 (before), 1363.8 (after)
 !  . Memory usage      : 6.0 Mb (before), 8.4 Mb (after)
 ! ----------------------------------------------------------------------------
 !   Branches  Non-fixed                Branch decision                   Best
          50        172               where#41 !=   32
         100        126              where#100 !=   17
         150         88              where#101 !=   46
 *       166      0.14s               where#87  =   17                       0
 ! Search terminated, replaying optimal solution
 ! ----------------------------------------------------------------------------
 ! Solution status          : Terminated normally, optimal found (tol. = 0)
 ! Number of branches       : 166
 ! Number of fails          : 35
 ! Total memory usage       : 9.6 Mb (9.4 Mb CP Optimizer + 0.2 Mb Concert)
 ! Time spent in solve      : 0.16s (0.16s engine + 0.00s extraction)
 ! Search speed (br. / s)  : 1062.4
 ! ----------------------------------------------------------------------------
```

Engine Log for a CP model (`steelmill` project)

Statistics

The **Statistics** tab shows details of the algorithm used by the solving engine.

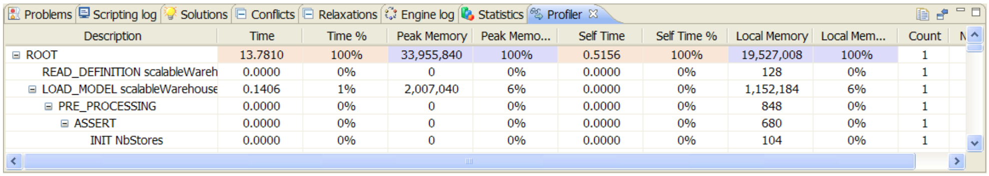


Statistic	Value
⊟ Cplex	
Constraints	250
Variables	10051
Non-zero coefficients	20050
⊟ MIP	
Objective	1479.857142857143
Incumbent	1480.0
Nodes	2075
Remaining nodes	1
Iterations	18852
⊟ Solution pool	
Count	6
Mean objective	3000.0

Statistics for an MP model (scalable configuration of `warehouse` project)

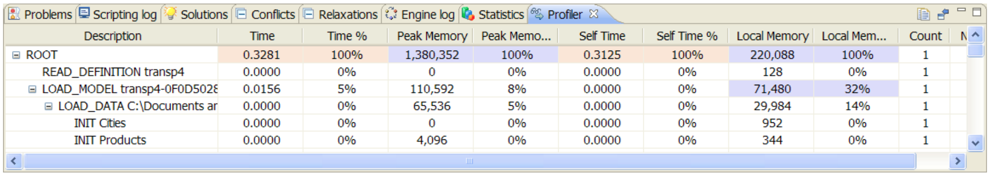
Profiler

The profiling tool computes the time and memory used by each execution step listed in the **Description** tree on the right and displays it as a table in the **Profiler** tab of the Output Area. You can use this information to improve the model so that it executes faster and consumes less memory. The Profiler table also displays details of model extraction and engine search during the solving phase. See *Profiling the execution of a model* in *IDE Tutorials*.



Description	Time	Time %	Peak Memory	Peak Memo...	Self Time	Self Time %	Local Memory	Local Mem...	Count	N
⊟ ROOT	13.7810	100%	33,955,840	100%	0.5156	100%	19,527,008	100%	1	
READ_DEFINITION scalableWareh	0.0000	0%	0	0%	0.0000	0%	128	0%	1	
⊟ LOAD_MODEL scalableWarehouse	0.1406	1%	2,007,040	6%	0.0000	0%	1,152,184	6%	1	
⊟ PRE_PROCESSING	0.0000	0%	0	0%	0.0000	0%	848	0%	1	
⊟ ASSERT	0.0000	0%	0	0%	0.0000	0%	680	0%	1	
INIT NbStores	0.0000	0%	0	0%	0.0000	0%	104	0%	1	

Profiler table for an MP model (scalable configuration of `warehouse` project)



Description	Time	Time %	Peak Memory	Peak Memo...	Self Time	Self Time %	Local Memory	Local Mem...	Count	N
⊟ ROOT	0.3281	100%	1,380,352	100%	0.3125	100%	220,088	100%	1	
READ_DEFINITION transp4	0.0000	0%	0	0%	0.0000	0%	128	0%	1	
⊟ LOAD_MODEL transp4-0F0D5028	0.0156	5%	110,592	8%	0.0000	0%	71,480	32%	1	
⊟ LOAD_DATA C:\Documents ar	0.0000	0%	65,536	5%	0.0000	0%	29,984	14%	1	
INIT Cities	0.0000	0%	0	0%	0.0000	0%	952	0%	1	
INIT Products	0.0000	0%	4,096	0%	0.0000	0%	344	0%	1	

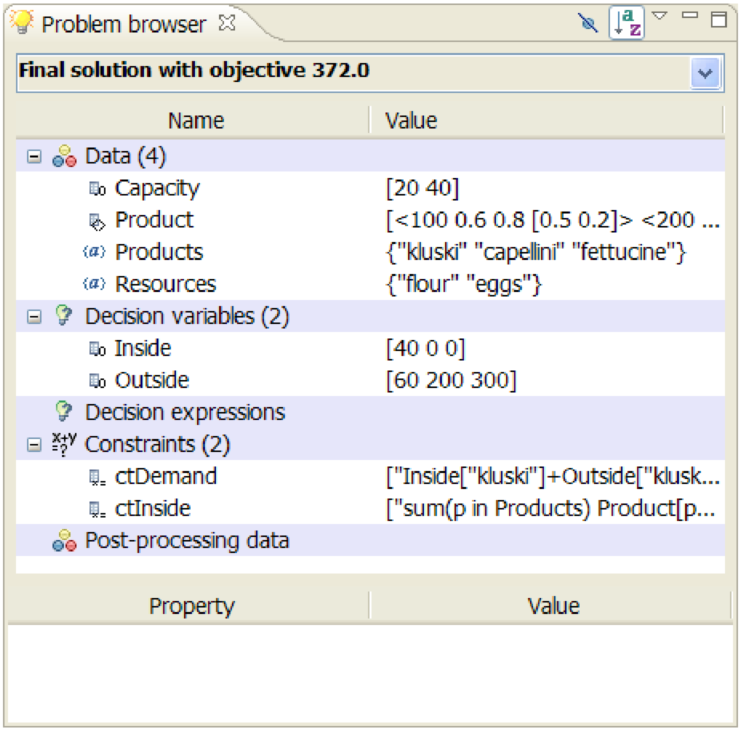
Profiler table for a CP model (`transp4.mod`)

Understanding the Problem Browser

The Problem Browser shows a structured view of the problem expressed by the model. See *The Problem browser* in the *IDE Reference* and *Doing more with the Problem Browser* in the *IDE Reference* for a complete presentation.

When you first open a project, the Problem Browser is empty. You can use it to browse the model before any execution and to examine the values after execution.

When you execute a project, the Problem Browser provides another way for you to examine the solution to your model, in addition to what you see in *The Output tabs*. It summarizes information about the data structures defined in the model to express the optimization problem. After you have executed the **Using tuples** configuration of the `production` example, the Problem Browser displays values for model elements.



Final solution with objective 372.0

Name	Value
⊟ Data (4)	
Capacity	[20 40]
Product	[<100 0.6 0.8 [0.5 0.2]> <200 ...
Products	{"kluski" "capellini" "fettucine"}
Resources	{"flour" "eggs"}
⊟ Decision variables (2)	
Inside	[40 0 0]
Outside	[60 200 300]
Decision expressions	
⊟ Constraints (2)	
ctDemand	["Inside["kluski"]+Outside["klusk...
ctInside	["sum(p in Products) Product[p...
Post-processing data	

Property	Value

Problem Browser after execution (`product.mod`)

Observe the Problem Browser window.

- The drop-down list at the top displays the final solution, which is the only solution it contains after the execution of the basic configuration of the `production` example. For run configurations that generate more than one solution, the list displays the solution pools that were computed by the engine. Selecting one of these solutions from the drop-down list displays data for that solution in the lower part of the Problem Browser. See *Working with the solution pool* in *IDE Tutorials*.

♦ The categories in the **Name** column (Data, Decision Variables, etc.) are populated with model objects and expanded to show the corresponding values. The order within each category is alphabetical.

♦ The **Value** column shows values for some of the model objects. The Property name and Property value columns remain empty until you select a model element.

For a large tuple set, the values may not all be visible within the window. In this case, an ellipsis appears at the end of the cell. Pass the cursor over the column to display all the values in a tooltip.

Note: If the model has only unlabeled constraints, the **Constraints** line is empty. To observe this, comment out the constraint labels and execute the project again. See *Constraint labels* in the *Language Reference Manual*.

Index