



# **IBM ILOG JViews Diagrammer V8.6**

## **Using the Dashboard Editor**

# Copyright

## Copyright notice

© Copyright International Business Machines Corporation 1987, 2009.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

## Trademarks

IBM, the IBM logo, ibm.com, WebSphere, ILOG, the ILOG design, and CPLEX are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

## Notices

For further copyright information see `<installdir>/license/notices.txt`.

## *Table of contents*

|  |           |
|--|-----------|
| <b>What is a dashboard.....</b>                              | <b>5</b>  |
| <b>Using the Dashboard Editor.....</b>                       | <b>7</b>  |
| <b>Running the Dashboard Editor.....</b>                     | <b>9</b>  |
| <b>Creating a dashboard.....</b>                             | <b>11</b> |
| <b>Adding a symbol to a dashboard.....</b>                   | <b>12</b> |
| <b>Linking symbols.....</b>                                  | <b>13</b> |
| Adding links.....  | 14        |
| Link-in and link-out ports.....                              | 16        |
| Enabling link layout.....                                    | 17        |
| <b>Editing the dashboard background.....</b>                 | <b>19</b> |
| Adding shape background objects.....                         | 20        |
| Adding and editing polyline objects.....                     | 21        |
| Adding a point to a polyline object.....                     | 22        |
| Editing background object properties.....                    | 23        |
| <b>Using layers in a dashboard diagram.....</b>              | <b>25</b> |
| Understanding layers.....                                    | 26        |
| Setting the default insertion layer.....                     | 28        |
| Raising and lowering a background object between layers..... | 29        |
| Raising and lowering a layer.....                            | 30        |
| Setting layer visibility.....                                | 31        |
| Setting layer transparency.....                              | 32        |

|  |           |
|--|-----------|
| <b>Saving a dashboard.....</b>                                     | <b>33</b> |
| <b>Changing symbol parameters.....</b>                             | <b>34</b> |
| <b>Aligning and grouping objects.....</b>                          | <b>36</b> |
| <b>Testing your dashboard.....</b>                                 | <b>37</b> |
| <b>Dashboard XML files.....</b>                                    | <b>38</b> |
| <b>Loading a palette.....</b>                                      | <b>40</b> |
| <b>Adding charts.....</b>  | <b>41</b> |
| <b>Opening a dashboard.....</b>                                    | <b>42</b> |
| <b>Editing multiple documents.....</b>                             | <b>43</b> |
| <b>Opening a dashboard from within a dashboard.....</b>            | <b>44</b> |
| <b>Programming the Dashboard Editor.....</b>                       | <b>47</b> |
| <b>A basic Dashboard Editor.....</b>                               | <b>49</b> |
| <b>The Dashboard Editor sample.....</b>                            | <b>51</b> |
| <b>IlvDashboardDiagram and IlvDashboardEditor.....</b>             | <b>52</b> |
| <b>Dashboard GUI components.....</b>                               | <b>55</b> |
| Adding menus.....  | 56        |
| Adding toolbars.....   | 57        |
| Adding the Tree pane.....  | 59        |
| Adding a Property pane.....  | 60        |
| Using the Overview pane.....                                       | 61        |
| Using the Palettes pane.....                                       | 62        |
| <b>Dashboard actions.....</b>                                      | <b>64</b> |
| Retrieving the link layout of a dashboard.....                     | 66        |
| Reading and writing dashboards.....                                | 67        |
| Changing the value of dashboard symbol parameters dynamically..... | 68        |
| <b>Index.....</b>  | <b>71</b> |

---

## What is a dashboard

A dashboard diagram is a graphical component composed of IBM® ILOG® JViews symbols arranged on an editable background. It is used to display business or system critical information graphically.

The JViews Diagrammer Dashboard Editor sample application shows how to implement a Dashboard Editor application programmatically. Dashboard functionality is implemented in the `ilog.views.dashboard` package. It is used to:

- ◆ Load and animate a dashboard diagram programmatically.
- ◆ Create your own graphical dashboard diagram editor applications.



# *Using the Dashboard Editor*

Shows how to use the Dashboard Editor.

## **In this section**

### **Running the Dashboard Editor**

Explains how to launch the Dashboard Editor under Windows® and under UNIX® .

### **Creating a dashboard**

Shows how straightforward it is to create a dashboard document.

### **Adding a symbol to a dashboard**

Shows how to add your first symbol to the dashboard.

### **Linking symbols**

Describes the linking mechanism in a dashboard diagram.

### **Editing the dashboard background**

Shows you how to add and edit background objects in a dashboard diagram.

### **Using layers in a dashboard diagram**

Shows how dashboard diagrams use a system of layers to organize the way symbols, links and background objects are displayed.

### **Saving a dashboard**

Explains the save options of a dashboard diagram.

### **Changing symbol parameters**

Explains how to change the value of a symbol parameter.

**Aligning and grouping objects**

Explains how to group background object and align symbols and background objects.

**Testing your dashboard**

Explains how to display a preview of your dashboard.

**Dashboard XML files**

Illustrates the structure of a dashboard XML file.

**Loading a palette**

Explains how to load a palette in the Dashboard Editor.

**Adding charts**

Explains how to add charts to a dashboard.

**Opening a dashboard**

Explains how to open a dashboard file.

**Editing multiple documents**

Explains how to edit several dashboard files.

**Opening a dashboard from within a dashboard**

Explains how to link two dashboards to open one from within the other.



---

## Running the Dashboard Editor

IBM® ILOG® JViews Dashboard Editor lets you edit and display dashboard diagrams made from JViews symbols.

You can run the Dashboard Editor under the Windows® or UNIX® operating systems. The startup conditions are the same in both cases.

### To run the Dashboard Editor under Windows:

- ◆ In the Start menu, select **All Programs>IBM ILOG>IBM ILOG JViews Diagrammer 8.6>Dashboard Editor**.

IBM® ILOG® is the default program group and may be different if you have installed JViews Diagrammer into a different program group.

-- or --

1. Go to the directory `<installdir>/jviews-diagrammer86/samples/dashboard/dashboardeditor`.
2. Double-click `run.bat`.

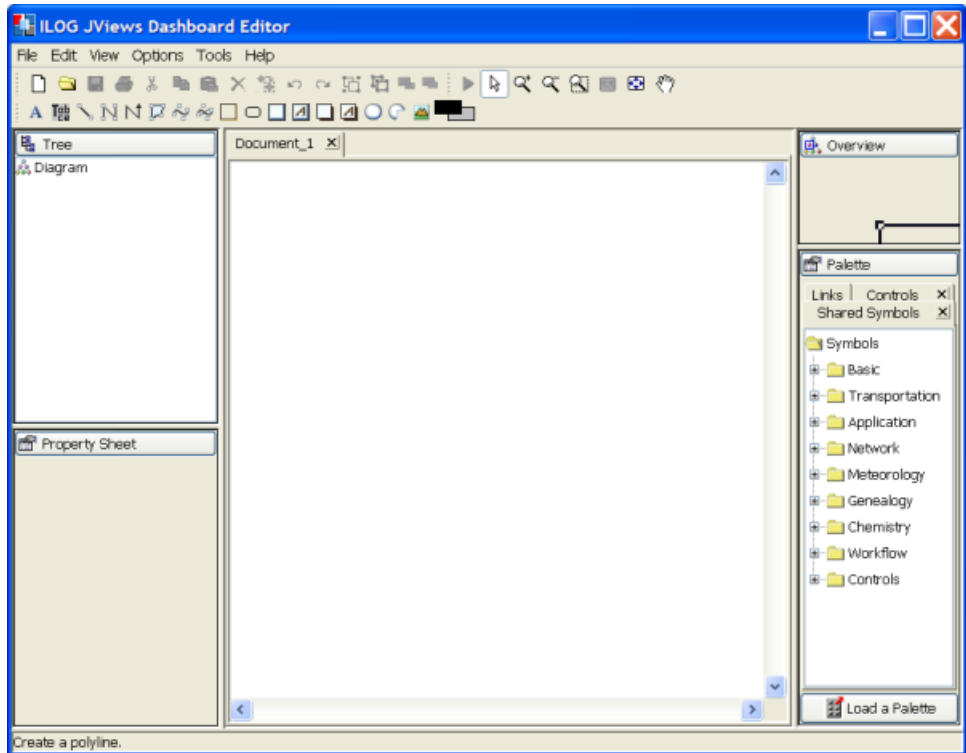
### To run the Dashboard Editor under UNIX:

1. Go to the directory `<installdir>/jviews-diagrammer86/samples/dashboard/dashboardeditor`.

Make sure your PATH environment variable is set correctly to find this directory.

2. Enter `run.sh`.

The Dashboard Editor opens.



### *The Dashboard Editor*

The Dashboard Editor window includes the following panes:

- ◆ Overview: displays the part of the dashboard visible in the drawing window.
- ◆ Tree: displays the list of the symbols contained in the dashboard diagram.
- ◆ Property Sheet: displays the parameters attached to the symbol or the properties of the background object currently selected.
- ◆ Palette: displays the symbol palettes open in Dashboard Editor.
- ◆ Editor: a graphical component used for creating and editing dashboard diagrams.

---

## Creating a dashboard

When you open Dashboard Editor, the following objects are opened by default:

- ◆ A first dashboard diagram, `Document_1`, in the editor pane in the center of the interface.
- ◆ Four example symbol palettes in the Palette pane at the right of the interface.

For more information about palettes and symbols, see [Using the Symbol Editor](#).

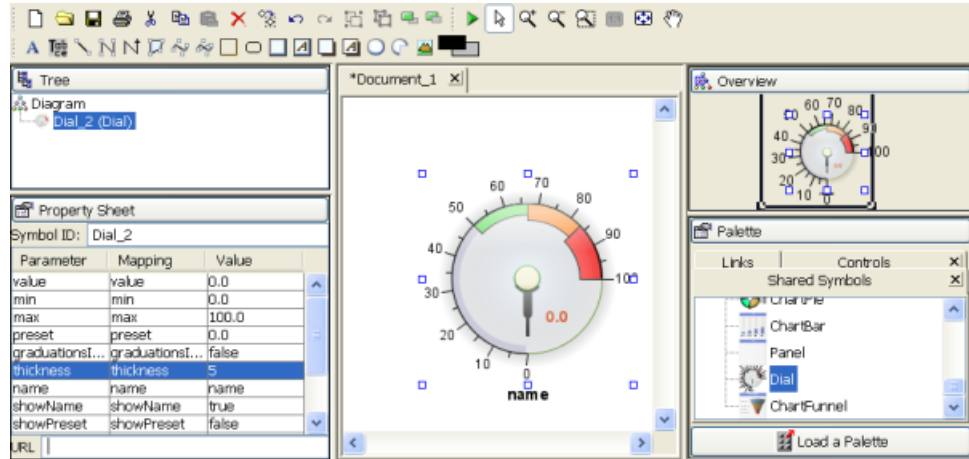
Although it contains no symbols, `Document_1` is an active dashboard diagram. You can add symbols and background objects directly into this diagram.

## Adding a symbol to a dashboard

You add a symbol to a dashboard diagram by dragging a symbol from the Dashboard Editor Palettes pane into the dashboard drawing pane.

- ◆ Drag the Dial symbol from the Controls category in the SharedSymbols palette pane to the `Document_1` drawing window.

The result is shown in the following figure.



*A symbol in Dashboard Editor*

For more information about the data displayed in the dashboard panes, see *Dashboard GUI components*.

# *Linking symbols*

Describes the linking mechanism in a dashboard diagram.

## **In this section**

### **Adding links**

Explains how to link two symbols in a dashboard diagram.

### **Link-in and link-out ports**

Describes how you can specify the exact connection points of the links on a symbol.

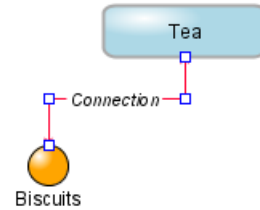
### **Enabling link layout**

Explains how to apply an automatic link layout in a dashboard diagram.

## Adding links

A link is a special kind of symbol used to connect two standard symbols. You can customize the shape, color, visibility, orientation, and associated text and shortcut menus for Labelled links. For less sophisticated links, such as the Simple Link or the Colored Link, you can change a limited number of these parameters.

| Parameter       | Mapping        | Value          |
|-----------------|----------------|----------------|
| orthogonal      | orthogonal     | true           |
| color           | color          |                |
| oriented        | oriented       | true           |
| backOriented    | backOriented   | true           |
| label           | label          | Connection     |
| labelForeground | labelForegr... |                |
| labelBackground | labelBackgr... |                |
| visible         |                | true           |
| toolTipText     |                | Tea and Bis... |
| popupMenuName   |                |                |
| sensitive       |                | true           |



### *Link parameters*

#### **To link two symbols in a dashboard diagram:**

1. Select a link from the Links tab in the Palettes pane.
2. Drag the link from the Palettes pane to the first symbol you want to connect.
3. Click the second symbol.

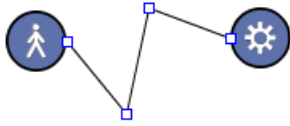


### *Adding a link between two symbols*

#### **To add intermediate points on the link:**

1. Select the link.
2. Press the CTRL key.
3. Click the link to add new points.

You can then move the link points inside the diagram.



*Adding intermediate points to a link*

**To remove intermediate points on a link:**

1. Press the CTRL key.
2. Click the intermediate point you want to remove.

---

## Link-in and link-out ports

By default, as a symbol moves in a dashboard diagram, the link between two symbols connects to the closest ports.



### *Standard links attach to the closest port on a symbol*

A palette symbol can be designed with elements that can be used as link-in or link-out ports. Links connected to link-in and link-out ports stay bound to these ports as the different symbols move.

Instead of connecting a link to the base shape of a symbol, you can connect it to a specific element of the symbol defined as a link port. It is in the Symbol Editor that you can define a symbol element as an in port, an out port, or an in and out port. See *Using link-in and link-out ports* in *Using the Symbol Editor* for more information.

When you choose the source symbol of a link, if the mouse pointer is on a link-out port, this symbol element is highlighted. When you choose the target symbol of a link, if the mouse pointer is on a link-in port, this symbol element is highlighted.



---

## Enabling link layout

The Dashboard Editor allows you to perform an automatic link layout of your dashboard diagram.

### To perform an automatic link layout:

- ◆ Choose **Options>Link Layout Enabled** in the menu bar.

A check mark appears in front of the menu item to indicate that the layout is enabled.

The intermediate points of your links, if any, are replaced by the points computed by the JViews link layout. As long as the option is enabled, the links are automatically adjusted when you move a node in the dashboard diagram.

If you disable the link layout, the intermediate points of your links remain fixed unless you explicitly move them.

The Link Layout Enabled option is saved with the dashboard file. If you open a file in which the option is enabled, the link layout will automatically be activated.

If you do not need to move your symbols at run time, you may want to disable the Link Layout Enabled option before saving the file.

For more information on the link layout, see Link layout (LL) in the *Using Graph Layout Algorithms* user's documentation.



# *Editing the dashboard background*

Shows you how to add and edit background objects in a dashboard diagram.

## **In this section**

### **Adding shape background objects**

Explains how to add a background to the dashboard.

### **Adding and editing polyline objects**

Explains how to add a polyline object to a dashboard.

### **Adding a point to a polyline object**

Explains how to add a point to a polyline.

### **Editing background object properties**

Explains how to edit properties of the background objects.

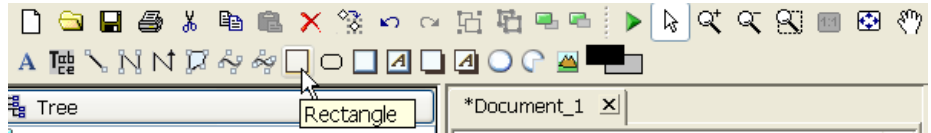
## Adding shape background objects

**To add a shape background object to a dashboard diagram:**

1. Select the object in the background editing toolbar.
2. Drag the mouse over the area where you want to place the background object.

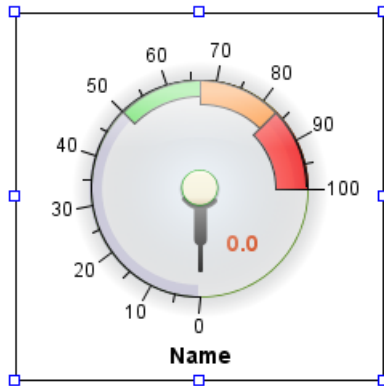
**To add a frame around the dashboard symbol you created in *Adding a symbol to a dashboard*:**

1. Select the Rectangle button in the background editing toolbar.



*The Rectangle button*

2. Drag the mouse over the Dial\_1 symbol in Document\_1.

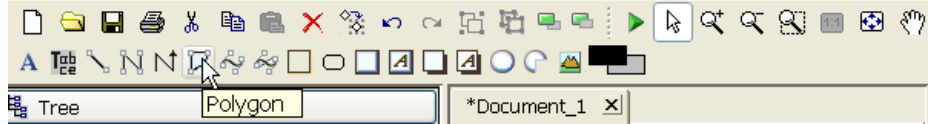


*A framed symbol in a dashboard diagram*

# Adding and editing polyline objects

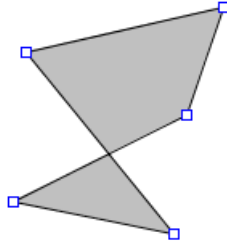
**To add a polyline object to a dashboard diagram:**

1. In the background editing bar, select the button for the polyline you want to add.  
You have the choice between a Polyline, Arrow Polyline, or Polygon object.



*Add a polygon to a dashboard diagram*

2. Click in the current diagram to mark each point in the polyline object.
3. Double-click in the diagram to mark the last point in the polyline object.



*A polygon*

---

## Adding a point to a polyline object

### To add a point to a polyline object:

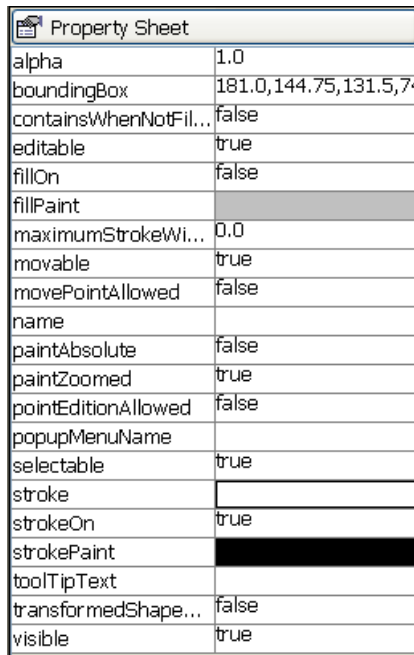
1. Select the polyline object you want to edit in current dashboard diagram.
2. Hold down the CTRL key and click inside the polyline object.

**Note:** To remove a polyline object point, press CTRL and click the existing points inside the polyline object.

## Editing background object properties

When a background object is selected, its properties are displayed in the Property Sheet pane. When you select more than one background object, only the properties common to the selected objects are displayed.

The following figure shows the properties for the frame you created in *Adding shape background objects*.



| Property Sheet        |                       |
|-----------------------|-----------------------|
| alpha                 | 1.0                   |
| boundingBox           | 181.0,144.75,131.5,74 |
| containsWhenNotFil... | false                 |
| editable              | true                  |
| fillOn                | false                 |
| fillPaint             |                       |
| maximumStrokeWi...    | 0.0                   |
| movable               | true                  |
| movePointAllowed      | false                 |
| name                  |                       |
| paintAbsolute         | false                 |
| paintZoomed           | true                  |
| pointEditionAllowed   | false                 |
| popupMenuName         |                       |
| selectable            | true                  |
| stroke                |                       |
| strokeOn              | true                  |
| strokePaint           |                       |
| toolTipText           |                       |
| transformedShape...   | false                 |
| visible               | true                  |

*Properties for a background object*

The property name is shown in the left column, its value in the right column. The value of each property can be edited.

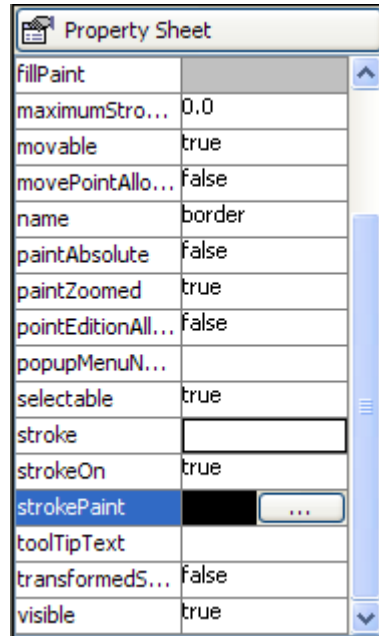
### To edit a property:

1. Click the corresponding cell in the right hand column.
2. Change the value accordingly.

### To change the line style of the frame you created in *Adding shape background objects*:

1. Select the frame you created in *Adding shape background objects*.
2. In the Property Sheet pane, click the cell to the right of the name field.
3. Type `border`.
4. Press ENTER.

- In the Property Sheet pane, click the cell to the right of the strokePaint field.  
A button appears at the right of the field.



*The stroke paint button*

- Click the button at the right of the strokePaint field.  
The Paint Editor window appears.
  - Select a tone of red in the Swatches tab color chooser.
  - Click Apply.  
The frame is now red.
  - Scroll down the Property Sheet pane until you see the visible field.
  - Click the cell to the right of the visible field.  
A drop-down list appears.
  - Select true.
- You have now customized a background object.



# *Using layers in a dashboard diagram*

Shows how dashboard diagrams use a system of layers to organize the way symbols, links and background objects are displayed.

## **In this section**

### **Understanding layers**

Describes the allocation of layers in a dashboard diagram.

### **Setting the default insertion layer**

Explains how to set a layer to be the default layer for the insertion of new background objects.

### **Raising and lowering a background object between layers**

Explains how to move a background object to a different layer.

### **Raising and lowering a layer**

Explains how to move all the objects of a layer to a different layer.

### **Setting layer visibility**

Explains how to make a layer visible or invisible.

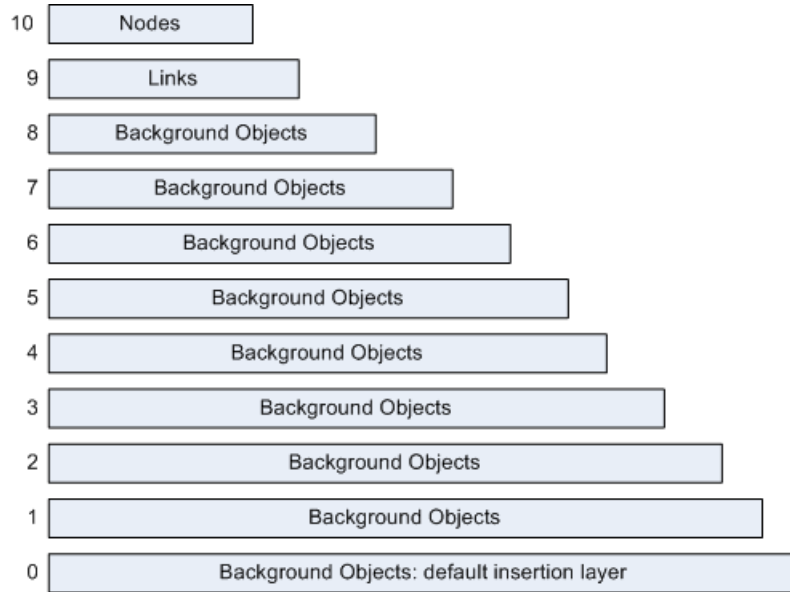
### **Setting layer transparency**

Explains how to adjust the transparency level of the objects on a layer.

---

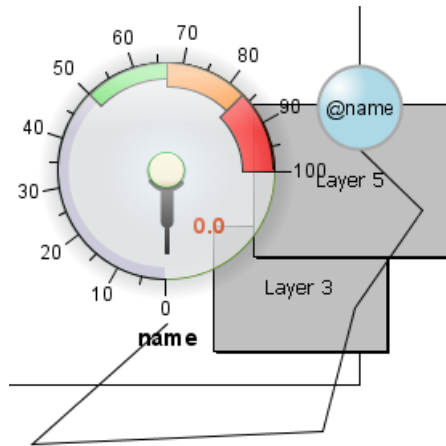
## Understanding layers

A dashboard diagram allocates eleven layers for symbols and background objects. Symbols are stored in the layer 10 (the node layer) and links are stored in layer 9. Layers 0 to 8 are available for background objects. The following figure, shows the logical structure for layers in a dashboard diagram.



*The layer structure in a dashboard diagram*

The following figure shows that objects on a higher layer overlay the objects below it. Links are displayed above background objects and symbols are contained in the topmost layer.



*Symbols, links and background objects on different layers*

**Note:** Symbols and links cannot change layers.

---

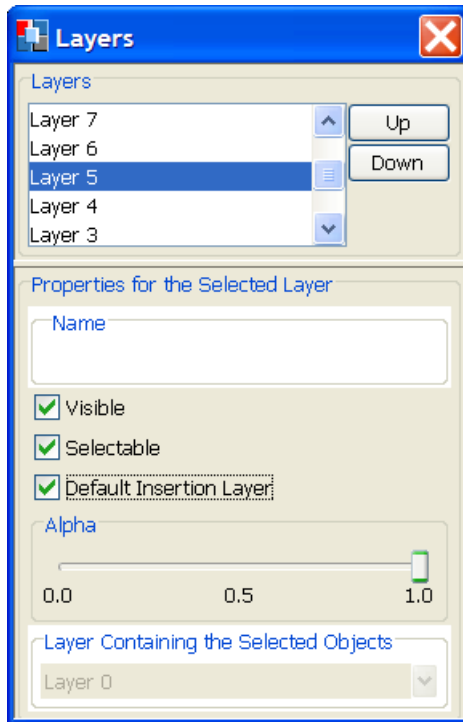
## Setting the default insertion layer

By setting the Default Insertion Layer in the Layers window, you can control into which layer the next background object added to the diagram will be inserted.

1. Open the Layers window by selecting **Tools>Layer Panel**.

The Layers window opens.

2. Select a layer in the Layers pane.
3. Select Default Insertion Layer.
4. Close the Layers window by clicking the red cross at the top right of the window.



*The Layers window*

---

## Raising and lowering a background object between layers

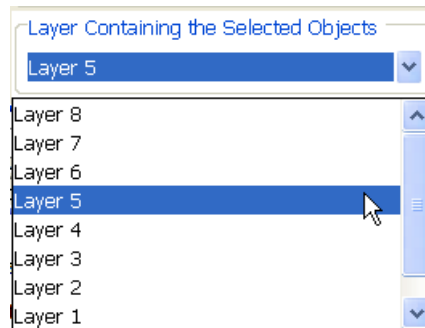
The Layer Containing the Selected Objects list, at the bottom of the Layers window, shows the layer in which the object currently selected in the dashboard diagram is stored.

### To raise or lower a background object between layers 0 and 8:

1. Select the background object you wish to move.
2. Select **Tools>Layer Panel**.

The Layers window opens.

3. Select the new layer for the object in the Layer Containing the Selected Objects list.



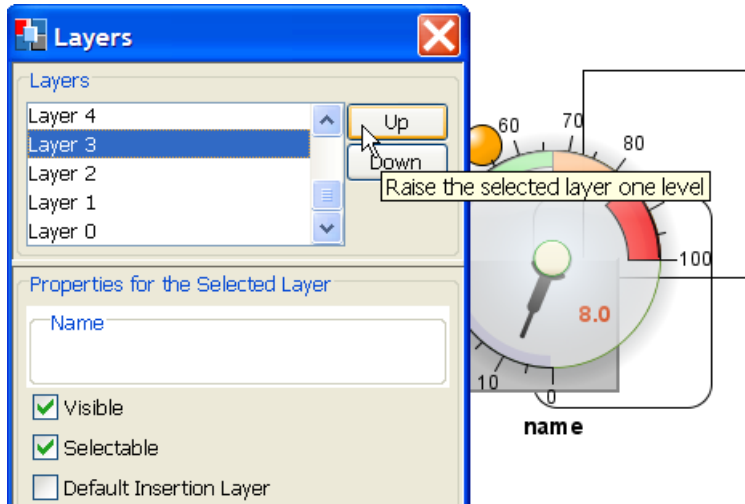
*Changing the layer of a background object*

4. Close the Layers window by clicking the close button.

## Raising and lowering a layer

A single layer can contain more than one background object. You can change the visibility of all objects in a layer simultaneously by raising or lowering the whole layer in the layer order.

1. Open the Layers window by selecting **Tools>Layer Panel**.
2. Select the layer you wish to raise or lower in the Layers list.
3. Click the Up or Down arrow to raise or lower the layer.



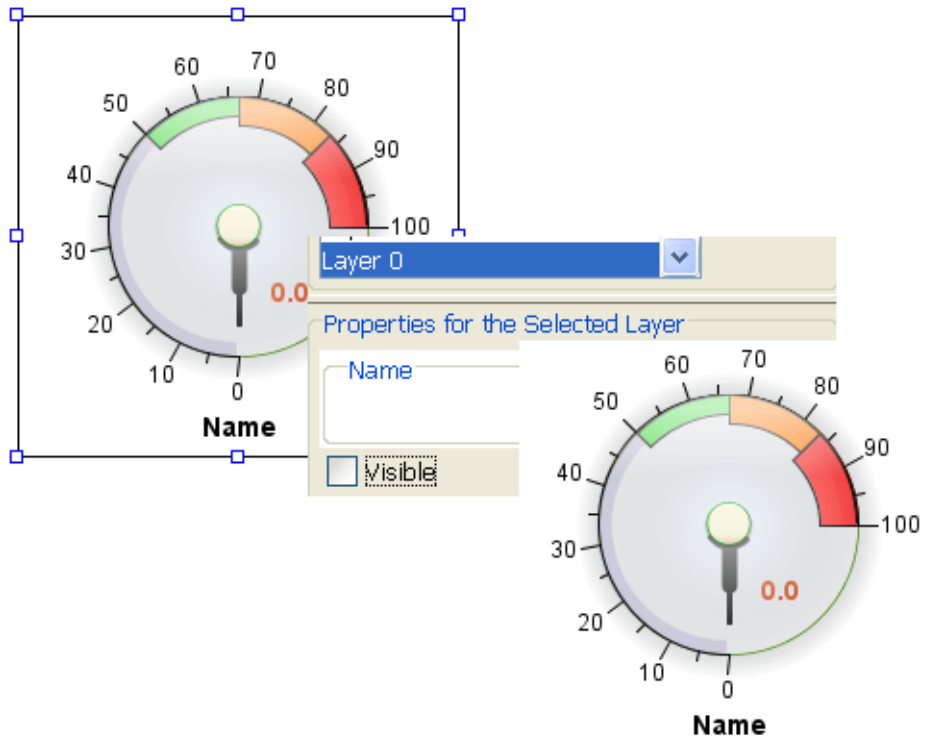
*Raising a layer in the dashboard diagram layer hierarchy*

## Setting layer visibility

Using the Layers window, you can make all objects stored in a specific layer visible or invisible.

### To make the contents of a layer invisible:

1. Open the Layers window.
2. Select the layer you wish to change in the Layers list at the top of the window.
3. Deselect Visible.
4. Close the Layers window.



*Setting a layer to be invisible*

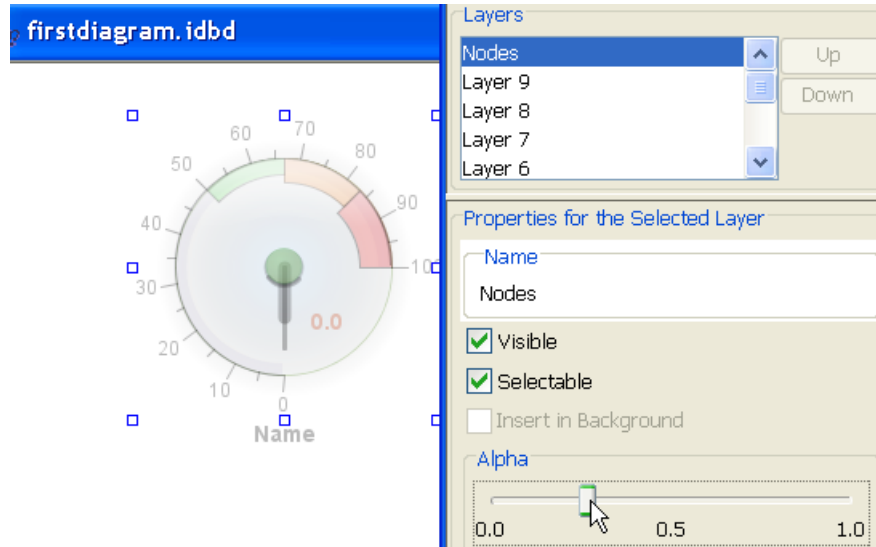
**Note:** You can also set the layer visibility by clicking the Up and Down arrows.

## Setting layer transparency

You can set the degree of transparency for all objects contained in a specific layer. Transparency changes can be applied to all layers, including layers 9 and 10 that contain symbols and links.

### To set the transparency level for a layer:

1. Open the Layers window.
2. Select the layer to change in the Layers pane at the top of the window.
3. Move the Alpha slider to the transparency level you require.



*Setting transparency for a layer*

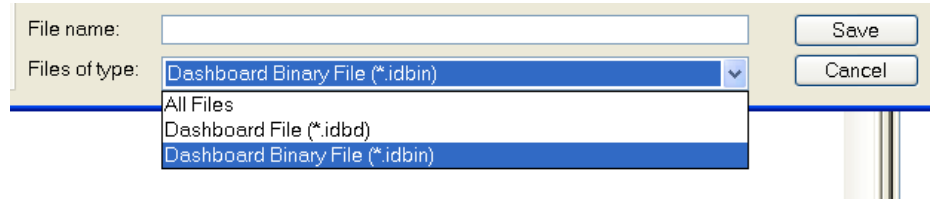


## Saving a dashboard

To save the dashboard diagram you created in *Creating a dashboard*:

1. Select **File>Save**.
2. Type `firstDashboard` in the File Name field.
3. Select the file format.

The Dashboard Editor supports 2 formats: XML (.idbd) and binary (.idbin).



### *Available dashboard file formats*

4. Click Save.

By default, IBM® ILOG® JViews dashboard diagram files are saved in binary format in the Dashboard Editor install directory. That is:

**<installdir>/jviews-diagrammer86/samples/dashboard/dashboardeditor.**

The binary format has the following qualities over the XML format.

- ◆ It is faster to save and load.
- ◆ The generated file is smaller.
- ◆ It needs less application memory when loading and saving the file.

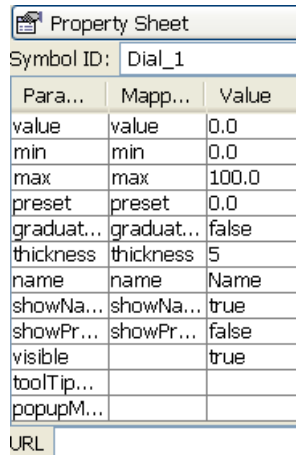
The dashboard XML format is more portable. If you need to share dashboard files with applications based on different versions of JViews Diagrammer, use the XML format.

A dashboard application or editor based on a newest version of JViews Diagrammer can load the binary files saved in previous versions of Dashboard Editor. However, a binary file saved by the latest version may not be loadable by previous versions of Dashboard Editor.

---

## Changing symbol parameters

When a symbol is selected, its parameters are displayed in the Property Sheet pane. When you select more than one symbol, only the parameters common to the selected symbols are displayed.



| Para...    | Mapp...    | Value |
|------------|------------|-------|
| value      | value      | 0.0   |
| min        | min        | 0.0   |
| max        | max        | 100.0 |
| preset     | preset     | 0.0   |
| graduat... | graduat... | false |
| thickness  | thickness  | 5     |
| name       | name       | Name  |
| showNa...  | showNa...  | true  |
| showPr...  | showPr...  | false |
| visible    |            | true  |
| toolTip... |            |       |
| popupM...  |            |       |

URL

*The Dashboard Editor property sheet*

A symbol has three property columns:

- ◆ **Parameter:** the name of the parameter as defined in the Symbol Editor.
- ◆ **Mapping:** the name of the model object property used to input data into the symbol.
- ◆ **Value:** the current value of the parameter.

### **To change the value of a parameter:**

1. Select the parameter to be changed in the Property Sheet pane.
2. Click the Value column for the parameter you wish to update.
3. Type the new value in the Value column.
4. Press ENTER.
5. Press CTRL+S to save the changes made to the dashboard diagram.

You see the symbol bound to the parameter change aspect in function of the changes to parameter values.

Change the `Mapping` field in the Property Sheet to the name of external variable used to change the value of a parameter.

For more information about parameters in symbols, see [Using the Symbol Editor](#).

For information about properties in the dashboard diagram API, see [Adding a Property pane](#).

**Note:** You can only set the value of symbol parameters in the Dashboard Editor. To edit symbols you must use IBM® ILOG® JViews Symbol Editor.

---

## Aligning and grouping objects

The Dashboard Editor provides the functionality to group background objects as well as to align symbols and background objects.

### **To group background objects in a dashboard diagram:**

1. Hold down the SHIFT key and select the objects you wish to group.
2. Click the Group Selection button in the edit toolbar.

### **To align objects in a dashboard diagram:**

1. Select the objects you wish to align.
2. Select **Edit>Align**, then click the alignment type you wish.

### **To align and group the objects in firstDashboard:**

1. Select the Dial\_1 symbol.
2. Hold down the SHIFT key and select the border background object.
3. Select **Edit>Align>Align Horizontal Center**.

**Note:** Grouping only works for background objects.

# Testing your dashboard

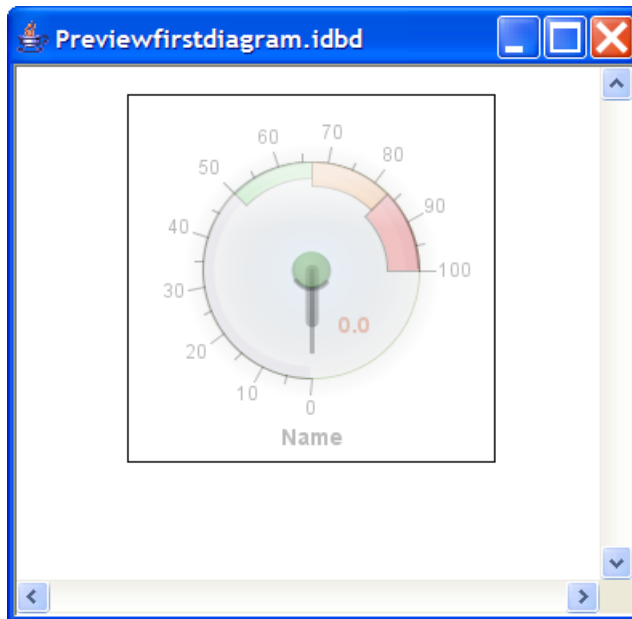
## To test your dashboard:

- ◆ Click the Preview button in the toolbar.



*The Preview button*

The Preview window opens with the current dashboard diagram loaded.



*The Preview window*

---

## Dashboard XML files

The Dashboard XML file structure contains the data necessary to render a dashboard diagram. Part of the contents of the `firstDashboard.idbd` file you created between *Creating a dashboard* and *Saving a dashboard* are shown below.

### The XML structure of a dashboard.idbd file

```
<Dashboard version="BC1">
  <symbols>
    <symbol id="Dial_1">
      <palette CSSResourceName="ilog/views/palettes/shared/symbols/Dial.css"

      className="Symbol" fullID="Symbols.Controls.Dial"
      paletteJarURL="file:/E:/ILOG/jviews-diagrammer86/lib/palettes/jviews-palette
      -shared-symbols-8.6.jar" palettePackage="ilog/views/palettes/shared/"
      paletteResourceName="ilog/views/palettes/shared/palette.xml"/>
      <geometry height="190.0" width="189.99998" x="162.57141" y="128.67432"/>
      <parameters>
        <parameter id="value" mapping="value" type="double">
          <value>0.0</value>
        </parameter>
        ...
        <parameter id="toolTipText" type="string"/>
        <parameter id="popupMenuName" type="string"/>
        <parameter id="sensitive" type="boolean">
          <value>true</value>
        </parameter>
      </parameters>
    </symbol>
  </symbols>
  <links/>
  <background>
    #ILOG Views Java 1.0
    #Reader : ilog.views.io.IlvGrapherReader
    ilog.views.IlvGrapher {
      Layers [
        ilog.views.IlvManagerLayer {
          index 0 visible true selectable true
          elements [ DEF obj_0 ilog.views.graphic.IlvGeneralPath {
            path "M51.0 23.0L281.0 23.0L281.0 251.5L51.0 251.5L51.0 23.0z"
            windingRule "nonzero"
            strokeOn true
            strokeColor 255 0 0
            fillOn false
            fillColor 255 0 51
          }
        ]
      } , ilog.views.IlvManagerLayer {
        index 1
        visible true
        selectable true
        elements [
          ]
        ]
      }
    ]
  }
</background>
</Dashboard>
```

```
    } ,  
    ...  
  }]]  
</background>  
</Dashboard>
```

A dashboard diagram file contains the following information:

- ◆ References to XML and CSS data used by the symbols included in the dashboard diagram. For each symbol, an `.idbd` file contains the following data:
  - The position of each symbol in the dashboard diagram.
  - Parameter data for each symbol in the dashboard diagram.
- ◆ The position and style of the background objects.

---

## Loading a palette

### To open a palette in Dashboard Editor:

1. Click the Load a Palette button at the bottom of the Palettes pane.  
The Open window is displayed.
2. Navigate to the directory containing your symbol palette files (.jar).  
By default, the Dashboard Editor stores palettes in the current user's home directory.
3. Select a palette file.
4. Click Open.

The symbols available in the palette are displayed in a new tab in the Palettes pane.



---

## Adding charts

To add a chart to a dashboard, use the predefined charts palette. This palette includes bar charts, area charts, pie charts, polyline charts, scatter charts, bubble charts, and variants of all these chart types.

The charts palette is only available when you have IBM® ILOG® JViews Charts installed, and a license for this product is present in your license key file. See Using License Keys for more information.

**Note:** If you do not have JViews Charts installed, chart symbols that you add to a dashboard are displayed as a text error message.

The chart symbols contained in the charts palette have names that indicate the kind of input data required. For example:

- ◆ The Polyline3 symbol represents three series of y values. Each series is given as an array of `double` values.
- ◆ The PolylineXY symbol represents a series of (x,y) points. The x and y values are given as separate arrays of `double` values.

Each series of y values or of (x,y) points can be accompanied with a list of labels. This list is specified as an array of `String` objects.

When you create and deploy an application that uses the charts palette, the main JViews Charts jar file, `jviews-chart.jar`, must be added to the CLASSPATH. If this is not done, Dashboard Editor will not be able to display the chart symbols.

**Note:** In JViews Diagrammer, the charts palette is available in Dashboard Editor only. It is not supported in the Symbol Editor nor in the Designer. You can use the predefined chart symbols in Dashboard Editor, however, you cannot create new chart symbols.

---

## Opening a dashboard

### To open an existing dashboard diagram:

1. In Dashboard Editor, select **File>Open**.

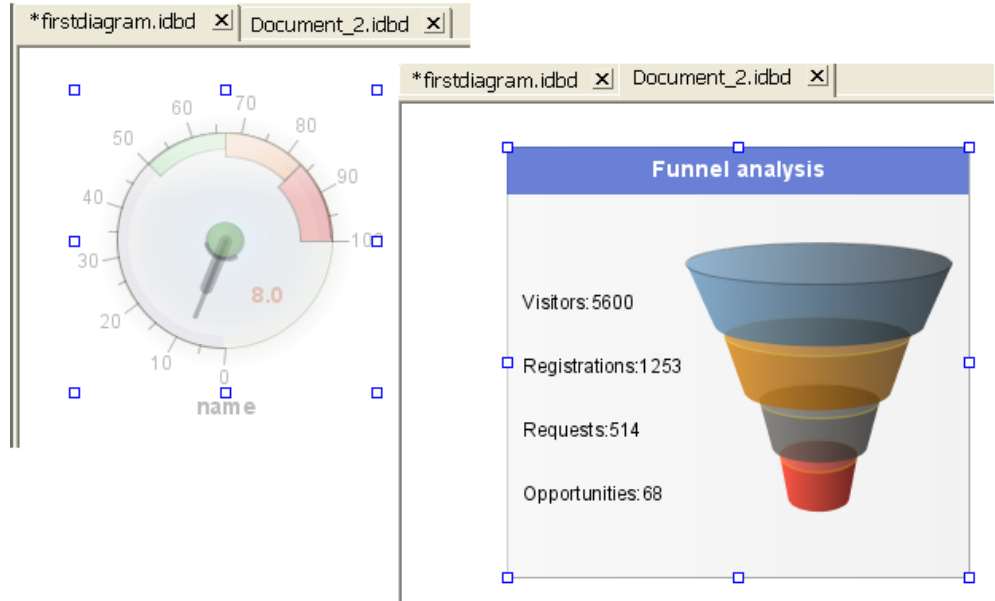
Files in both the binary and XML formats are displayed in the Open window.

2. Select a dashboard diagram file.
3. Click Open.

**Note:** When loading a dashboard diagram, the Dashboard Editor tries to locate and load missing symbol palettes. These palettes must be stored in a directory contained in the Java™ CLASSPATH for this to work correctly.

## Editing multiple documents

The Dashboard Editor lets you edit dashboard diagrams through a multidocument interface. This means you can add symbols from a single palette to multiple dashboard documents simultaneously.



*Editing multiple dashboard diagrams*

**To open multiple dashboard documents:**

- ◆ Select **File>New**.

-- or --

- ◆ Open an existing document as shown in *Opening a dashboard*.

## Opening a dashboard from within a dashboard

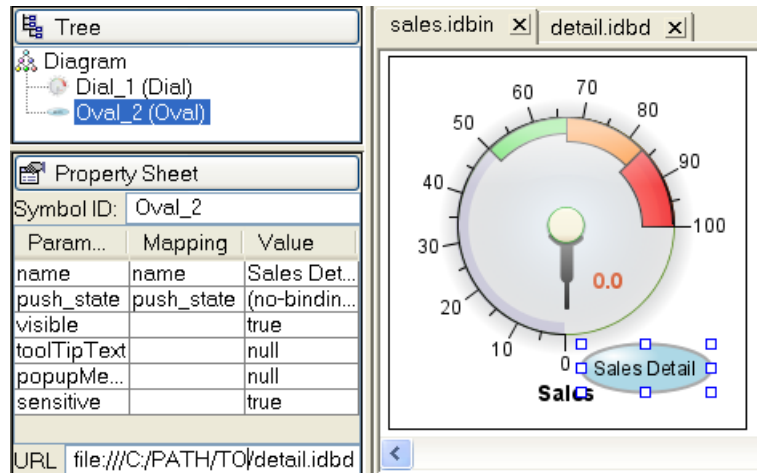
Dashboard Editor allows you to open external dashboards by linking them to a symbol contained in your original dashboard. For example, you have a dashboard that displays sales data and you would like to be able to open a dashboard that has a breakdown of the items sold and the sales made by staff. After creating both dashboards, you need to link a symbol from your sales dashboard to your detail dashboard. This is done using the URL property in the Parameters pane.

### To create a dashboard link:

1. Select the symbol in the sales dashboard you want to use as a link to the detail dashboard.
2. Write the URL to the dashboard in the URL field.

This can be either a fully qualified or a relative URL. For example:

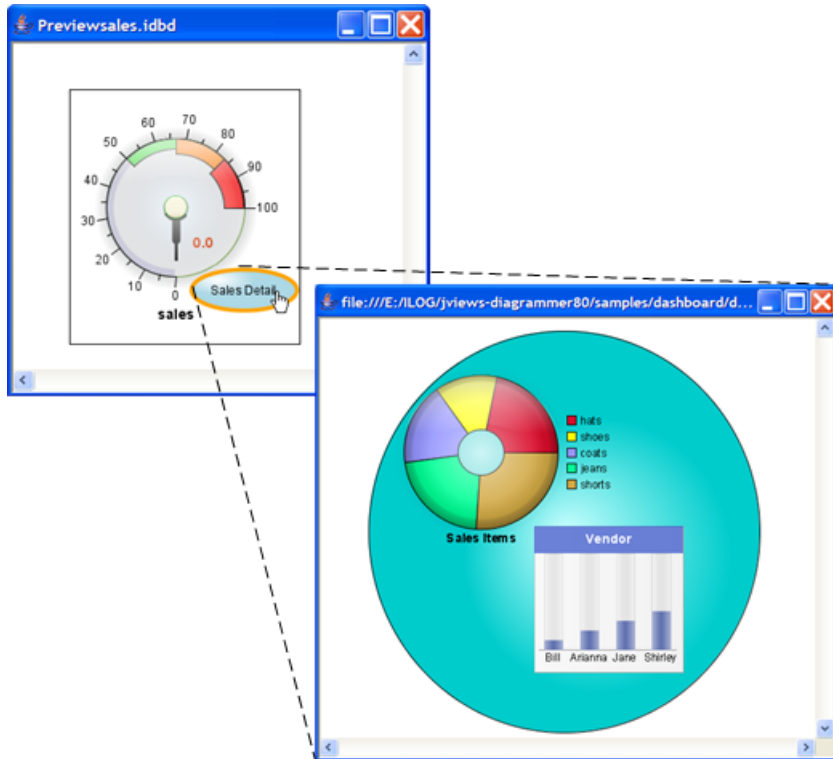
- ◆ `file:///C:/PATH/TO/detail.idbd`
- ◆ `./detail.idbd`



*Setting the URL for a dashboard*

3. Test you dashboard.

When you run your sales dashboard, clicking on the detail link will open the detail dashboard.



*Using the URL parameter to open another dashboard*



# *Programming the Dashboard Editor*

Shows how to integrate dashboard diagram facilities into a graphical application using the `ilog.views.dashboard` package.

## **In this section**

### **A basic Dashboard Editor**

Provides a code example to program a basic Dashboard Editor application.

### **The Dashboard Editor sample**

Presents the classes used to program a dashboard diagram.

### **IlvDashboardDiagram and IlvDashboardEditor**

Describes the main classes in a Dashboard Editor.

### **Dashboard GUI components**

Describes the classes and interfaces that help you ease the development of Swing GUIs containing one or more `IlvDashboardDiagram` objects. These classes are contained in the `ilog.views.dashboard` package

### **Dashboard actions**

Shows how to extend a dashboard application to have a new menu item.

### **Retrieving the link layout of a dashboard**

Shows how to return the link layout of a dashboard.

### **Reading and writing dashboards**

Explains how to read and write dashboards in two formats, `idbd` and `idbin`.

## **Changing the value of dashboard symbol parameters dynamically**

Explains how to add a dashboard to an application and change the parameter values to display dynamic data.



---

## A basic Dashboard Editor

The `ilog.views.dashboard` package is used to create dashboard diagram applications based on IBM® ILOG® JViews symbols and composite graphics. Extended from JViews Diagrammer code, the dashboard diagram package uses standard Swing components to make GUI applications.

The following example shows how to program a basic Dashboard Editor application.

### A basic dashboard editor application

```
import javax.swing.JFrame;
import java.awt.BorderLayout;
import java.net.URL;
import javax.swing.JFrame;
import javax.swing.ImageIcon;

import ilog.views.dashboard.IlvDashboardEditBar;
import ilog.views.dashboard.IlvDashboardEditor;
import ilog.views.dashboard.IlvDashboardSymbolPalette;
import ilog.views.dashboard.IlvDashboardDiagram;
import ilog.views.dashboard.IlvDashboardTabbedPane;
import ilog.views.diagrammer.application.IlvDiagrammerOverview;

public class DashEditorExample extends JFrame
{
    private IlvDashboardEditor dashEd;
    private IlvDashboardEditBar editToolBar;
    private IlvDashboardSymbolPalette palettePanel;
    private IlvDashboardDiagram dashDiag;

    public DashEditorExample(String[] args)
    {
        // Set up the application frame.
        super("Basic Diagrammer Application");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocation(100, 100);
        setSize(700, 700);

        // Create the editor with a single diagram.
        dashEd = new IlvDashboardEditor();
        dashEd.init(args);

        IlvDashboardTabbedPane tabPane = new IlvDashboardTabbedPane(dashEd);
        dashEd.setDashboardContainer(tabPane);

        // Add the Edit toolbar.
        editToolBar = new IlvDashboardEditBar();
        // Add the Symbol palette
        palettePanel = new IlvDashboardSymbolPalette(dashEd);

        // Add the Dashboard components into the JFrame
        getContentPane().setLayout(new BorderLayout());
    }
}
```

```
getContentPane().add(tabPane, BorderLayout.CENTER);
getContentPane().add(editToolBar, BorderLayout.NORTH);
getContentPane().add(palettePanel, BorderLayout.EAST);

dashEd.loadPalette("ilog/views/palettes/controls/");
dashEd.loadPalette("ilog/views/palettes/shared/");
dashEd.run();
}

public static void main(String[] args) {
    DashEditorExample basicEditor = new DashEditorExample(args);
    basicEditor.setVisible(true);
}
}
```

---

## The Dashboard Editor sample

For a complete example of how to program a dashboard diagram, see the source code of the Dashboard Editor sample in:

```
<installdir>/jviews-diagrammer86/samples/dashboard/dashboardeditor/src/  
DashboardEditor.java
```

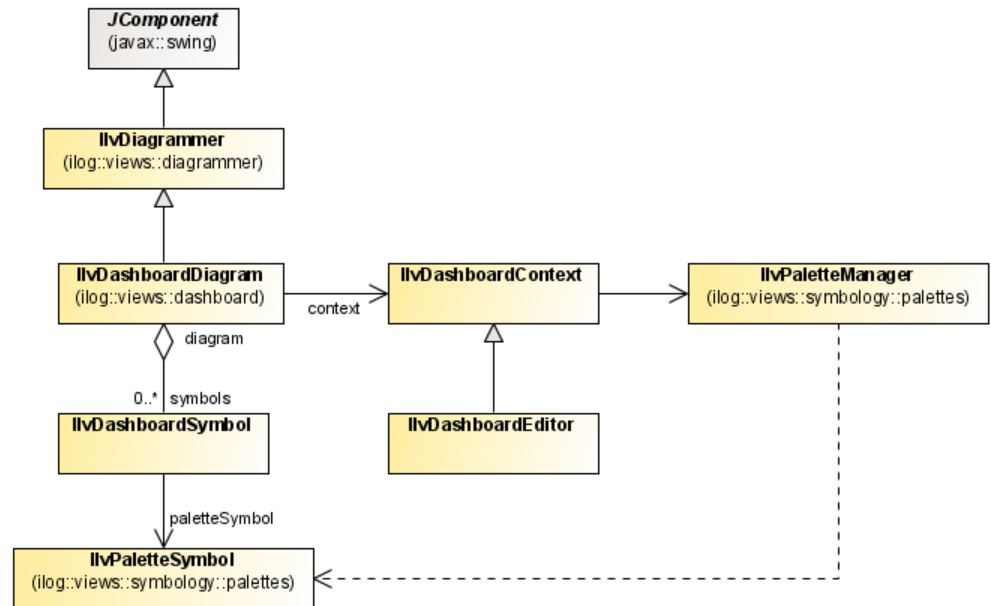
In the Dashboard Editor, an `IlvDashboardDiagram` is viewed and edited in an `IlvDashboardEditorFrame` object. An `IlvDashboardDiagram` object is a Swing component that displays a dashboard diagram composed of background objects and symbols. An `IlvDashboardEditorFrame` object displays multiple `IlvDashboardDiagram` instances and handles the palettes in which symbols are stored.

The `IlvDashboardDiagram` is derived from the `IlvDiagrammer` class. It uses dashboard-specific interactors, palettes, and style sheets.

## IlvDashboardDiagram and IlvDashboardEditor

The purpose of an `IlvDashboardDiagram` instance is to organize and edit symbols and background objects. Symbols are represented by one or more `IlvDashboardSymbol` objects. Background images are represented by standard IBM® ILOG® JViews graphic objects. That is, instances of `IlvGraphic` subclasses. Symbols and graphic objects can be added, removed, distributed and aligned using the user interface.

**Note:** Symbols cannot be grouped.



### *The main classes in Dashboard Editor*

`IlvDashboardDiagram` descends from `JComponent`. For this class to function properly, it needs to be placed in a containment hierarchy whose root is a top-level Swing container.

`IlvDashboardEditor` inherits from `IlvDashboardContext`. It serves two purposes:

- ◆ It handles all dashboard diagrams open in the Diagram Editor.
- ◆ It provides access to different resources needed by a dashboard application, such as symbol palettes.

For more information about palettes, see *Using the Palettes pane*.

The following code example shows how to create an editable dashboard diagram.

### An editable dashboard diagram

```
// Create the editor with a single diagram.
dashEd = new IlvDashboardEditor();
dashDiag = new IlvDashboardDiagram(dashEd);
...
// Set the Dashboard objects in the JFrame
getContentPane().setLayout(new BorderLayout());
getContentPane().add(dashDiag, BorderLayout.CENTER);
```



# *Dashboard GUI components*

Describes the classes and interfaces that help you ease the development of Swing GUIs containing one or more `IlvDashboardDiagram` objects. These classes are contained in the `ilog.views.dashboard` package

## **In this section**

### **Adding menus**

Shows how to set an `IlvDashboardMenuBar` in a Dashboard Editor application.

### **Adding toolbars**

Shows how to create a toolbar to edit a dashboard diagram.

### **Adding the Tree pane**

Shows how to add an `IlvDashboardTree` to a dashboard application.

### **Adding a Property pane**

Shows how to add an `IlvDashboardPropertyPanel` to a dashboard application.

### **Using the Overview pane**

Shows how to add an `IlvDiagrammerOverview` to a dashboard application.

### **Using the Palettes pane**

Shows how to add an `IlvDashboardSymbolPalette` to a dashboard application.

---

## Adding menus

The class `IlvDashboardMenuBar` is a Swing `JMenu` that accepts a set of `IlvDashboardAction` instances. It inherits from `IlvDiagrammerMenuBar` and includes File, Edit, View and Options menus.

The following code example shows how to set an `IlvDashboardMenuBar` in a Dashboard Editor application.

### Adding a menu bar to a Dashboard Editor application

```
dashEd = new IlvDashboardEditor();
IlvDashboardMenuBar menuBar = new IlvDashboardMenuBar(dashEd);
//set the menu for the JFrame
this.setJMenuBar(menuBar);
```

For an example of how to extend `IlvDashboardMenuBar` to add a new menu bar, see the Dashboard Editor sample. This sample can be found at:

**<installdir>/jviews-diagrammer86/samples/dashboard/dashboardeitor/src/DashboardEditor.java**



## Adding toolbars

The `ilog.views.dashboard` package implements two new toolbars, `IlvDashboardEditBar` and `IlvDashboardBackgroundBar`. Both are Swing `JToolBars`. They accept a set of `IlvDashboardAction` instances and contain predefined actions to control and edit a dashboard diagram.

`IlvDashboardEditBar` inherits from `IlvDiagrammerEditBar`. It controls editing, grouping and duplication functionality.



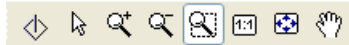
An `IlvDashboardEditBar`

`IlvDashboardBackgroundBar` inherits from `IlvGraphicPaletteBar`. It sends *Dashboard actions* to edit background objects in the dashboard diagram.



The `IlvDashboardBackgroundBar`

`IlvDashboardDiagram` inherits from `IlvDiagrammer`. You can use standard `Diagrammer` toolbars to control objects in a dashboard diagram. For example, `IlvDiagrammerViewBar` is used for zoom and pan operations.



An `IlvDiagrammerViewBar`

The following code example shows how to create a toolbar to edit a dashboard diagram.

### Add an edit toolbar

```
JPanel panel = new JPanel(new BorderLayout());

JPanel panell = new JPanel(new IlvBetterFlowLayout(FlowLayout.LEADING, 0, 0))
;
IlvDashboardEditBar editToolBar = new IlvDashboardEditBar();
panell.add(editToolBar);
IlvDiagrammerViewBar viewToolBar = new IlvDiagrammerViewBar();
panell.add(viewToolBar);
panel.add(panell, BorderLayout.NORTH);

final IlvDashboardBackgroundBar paletteToolBar =
    new IlvDashboardBackgroundBar(this);
panel.add(paletteToolBar, BorderLayout.SOUTH);

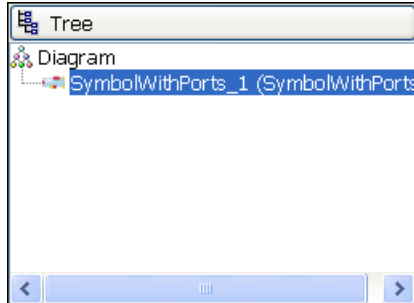
pcolors.setClient(new IlvPaletteColorSelector.Client() {
    public void backgroundSelected(Color color) {
        paletteToolBar.setPaletteBackground(color);
    }
    public void foregroundSelected(Color color) {
        paletteToolBar.setPaletteForeground(color);
    }
});
```

```
}  
});
```

---

## Adding the Tree pane

The Dashboard Editor uses `IlvDashboardTree` to display the order of the symbols in a dashboard diagram. `IlvDashboardTree` is a Swing `JTree`, it is used as an alternative way to view and select the objects in the diagram.



*An `IlvDashboardTree`*

**Note:** Background objects are not visible in the Tree pane. Only symbols and links are shown.

The following code example shows how to add an `IlvDashboardTree` to a Dashboard application.

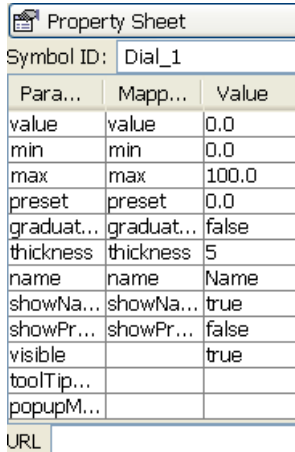
### Adding an `IlvDashboardTree` to a dashboard application

```
URL url = IlvDashboardTree.class.getResource("images/tree.gif");
ImageIcon icon = new ImageIcon(url);
IlvDashboardTree tree = new IlvDashboardTree();
JScrollPane treeScrollPane = new JScrollPane(tree);
IlvDashboardExpandablePane treeFrame = new IlvDashboardExpandablePane("Tree",
    icon, treeScrollPane);
```

---

## Adding a Property pane

The class `IlvDashboardPropertyPanel` contains a Swing `JTable` that displays the properties of objects selected in the diagram. Use the property sheet to view and edit the properties of a background object or the parameters of a symbol.



*An `IlvDiagrammerPropertyPanel`*

The following code example shows how to add an `IlvDashboardPropertyPanel` to a dashboard application.

### Adding an `IlvDashboardPropertyPanel` instance to a dashboard application

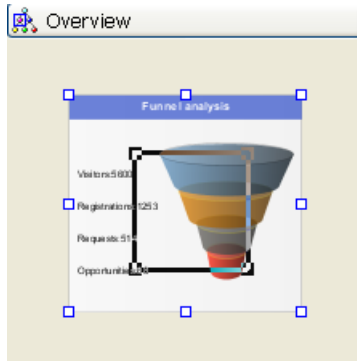
```
url = IlvDiagrammerPropertySheet.class.getResource("images/psheet.gif");
icon = new ImageIcon(url);
IlvDashboardPropertyPanel ppanel = new IlvDashboardPropertyPanel(this);

IlvDashboardExpandablePane psheetFrame =
    new IlvDashboardExpandablePane("Property Sheet", icon, ppanel);

IlvDashboardExpandableSplitPane split = new IlvDashboardExpandableSplitPane
(treeFrame, psheetFrame);
split.setResizeWeight(0.5);
split.setDividerLocation(300);
```

## Using the Overview pane

`IlvDashboardDiagram` inherits from `IlvDiagrammer`. You can use standard Diagrammer panes to control objects in a dashboard diagram. The Dashboard Editor uses `IlvDiagrammerOverview` to display a reduced view of a dashboard diagram. You can use it with the zoom facility of a diagram component to control which part of a large diagram is visible.



*An `IlvDiagrammerOverview`*

The following code example shows how to add an `IlvDiagrammerOverview` to a dashboard application.

### **Adding an `IlvDiagrammerOverview` instance to a dashboard application**

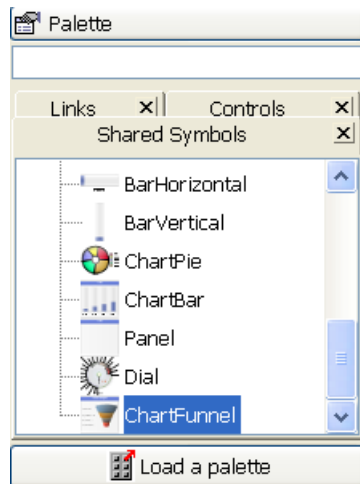
```
URL url = IlvDiagrammerOverview.class.getResource("images/overview.gif");
ImageIcon icon = new ImageIcon(url);
IlvDiagrammerOverview overview = new IlvDiagrammerOverview();

IlvDashboardExpandablePane overviewFrame = new IlvDashboardExpandablePane
("Overview", icon, overview);
```

## Using the Palettes pane

The class `IlvDashboardSymbolPalette` is a Swing `JPanel` that displays the symbols contained in the open palettes. Use the Palettes pane to do the following:

- ◆ View symbols in open palettes.
- ◆ Drag symbols from the Palettes pane into a dashboard diagram.



An `IlvDashboardSymbolPalette`

The following code example shows how to add an `IlvDashboardSymbolPalette` to a dashboard application.

### Adding a symbol palette to a dashboard application

```
url = IlvDiagrammerPropertySheet.class.getResource("images/psheet.gif");
icon = new ImageIcon(url);

IlvDashboardSymbolPalette palettePanel = new IlvDashboardSymbolPalette(this);
IlvDashboardExpandablePane paletteFrame =
    new IlvDashboardExpandablePane("Palette", icon, palettePanel);
try {
    loadPalette(new URL("file:data/palettes/palette-example.jar"));
    loadPalette(new URL("file:data/palettes/link.jar"));
    loadPalette(new URL("file:data/palettes/gauge.jar"));
} catch (Exception e) {
    e.printStackTrace();
}

IlvDashboardExpandableSplitPane split = new IlvDashboardExpandableSplitPane
(overviewFrame, paletteFrame);
split.setResizeWeight(0.75);
```

```
split.setDividerLocation(150);  
return split;
```

---

## Dashboard actions

`IlvDashboardAction` is a subclass of `IlvDiagrammerAction`. It inherits some actions available in a Diagrammer application. Some `IlvDiagrammerActions` are based on methods found in the `IlvDiagrammer` class. The redefinition of the corresponding methods in `IlvDashboardDiagram` changes the behavior of the inherited methods.

The following new actions have been added for dashboard diagrams:

- ◆ Importing a dashboard diagram file
- ◆ Sending the selected object to the top position in its layer
- ◆ Moving the selected object up one position in the stacking order of its layer
- ◆ Moving the selected object down one position in the stacking order of its layer
- ◆ Sending the selected object to the bottom position in its layer
- ◆ Saving the current dashboard document as a new file
- ◆ Saving the current dashboard document
- ◆ Opening an existing dashboard document

The following code example shows how to extend a dashboard application to have a new menu item.

### Extending a dashboard application

```
IlvDiagrammerAction.Handler userManualHandler =
    new IlvDiagrammerAction.Handler()
{
    public void perform(IlvDiagrammerAction action,
                      IlvDiagrammer diagrammer,
                      ActionEvent event)
        throws Exception {
        popUserManual();
    }

    public void update(IlvDiagrammerAction action,
                      IlvDiagrammer diagrammer)
        throws Exception {
        action.setEnabled(true);
    }
};
IlvDiagrammerAction.help.setHandler(userManualHandler);
...
public void popUserManual() {
    popBrowser(bundle.getString("Help.Menu.Help.File"),
              bundle.getString("Help.Menu.Help.Fallback.File"),
              bundle.getString("Help.Menu.Help.File.Missing"),
              bundle.getString("Help.Menu.Help.Failed"));
}
```



```
}  
}
```

---

## Retrieving the link layout of a dashboard

The following code shows how to return the link layout of a dashboard:

```
/**
 * Returns the link layout of a dashboard if it is installed.
 * @param dashboard The dashboard diagram.
 * @return The link layout, or null if it is not enabled.
 */
public static IlvGraphLayout getLinkLayout(IlvDashboardDiagram dashboard) {

    IlvGraphLayoutRenderer renderer =
        dashboard.getEngine().getLinkLayoutRenderer();
    if (renderer != null) {
        return renderer.getGraphLayout();
    }
    return null;
}
```

For this code to work correctly, link layout must be enabled. For information on how to do this, see *Enabling link layout*.

For more information on how to work with link layout using Java™ code, see Link layout (LL) in the *Using Graph Layout Algorithms* user's documentation.

---

## Reading and writing dashboards

Dashboards are saved in IBM® ILOG® JViews dashboard diagram (`.idbd`) and (`.idbin`) files.

The `.idbd` files are XML structures containing the data necessary to render the dashboard diagram. Dashboards are saved in XML format by calling `writeDashboard` and opened by calling `readDashboard`.

The `.idbin` are binary files that contain the dashboard diagram. Dashboards are saved in binary format by calling `writeBinary` or `writeBinary`. They are opened by calling `readBinary` or `readBinary`.

Binary files are smaller, faster to save and load, and need less application memory when loading and saving. The dashboard XML format is more portable if you need to share dashboard files with applications based on different versions of JViews Diagrammer.

---

### Using the compiled symbol Dashboard Reader

Instead of loading your dashboard binary file in an `IlvDashboardDiagram` object, you can load it in a simple `IlvGrapher` using the Compiled Symbol Dashboard Reader. See [Using compiled symbols in a runtime dashboard](#).

## Changing the value of dashboard symbol parameters dynamically

Once you have created your dashboard, you need to be able to add it to an application and change the parameter values to display dynamic data.

### To retrieve a dashboard symbol and change the value of a parameter:

- ◆ Write the following code.

The dashboard used is the one you created in *Adding a symbol to a dashboard*.

### Setting the values of dashboard symbols dynamically

```
public class dashboardUpdater extends JApplet implements Runnable
{
    private IlvDashboardDiagram _dashboard;

    public void init()
    {
        // create the dashboard.
        IlvDashboardContext context = new IlvDashboardContext();
        _dashboard = new IlvDashboardDiagram(context);
        try {
            // Load the dashboard diagram.
            url = new URL("file:./" + <path to dashboard>);
            _dashboard.readDashboard(url);
        } catch (IlvDiagrammerException er) {
            er.getCause().printStackTrace();
            System.exit(-1);
        } catch (Exception e) {
            ...
        }

        getContentPane().add(_dashboard);
    }

    ...

    public void updateDashBoard() {
        _dashboard.setAdjusting(true);
        Object dial1 = _dashboard.getObject("Dial_1");
        _dashboard.setObjectProperty(dial1, "value", 8);
        _dashboard.setAdjusting(false);
    }

    public void run()
    {
        ...
        updateDashBoard();
        ...
    }

    public static void main(String[] args) {
        dashboardUpdater dashboard = new dashboardUpdater();
    }
}
```

```
    dashboard.init();  
    dashboard.run();  
  }  
}
```

The value passed as third argument to the `setObjectProperty(java.lang.Object, java.lang.String, java.lang.Object)` method must have the right type for the symbol parameter given as second argument.

Symbols in the charts palette as shown in *Adding charts*, have array parameter types.

**To replace an entire array with values:**

- ◆ Pass the new array to the `setObjectProperty` method.

**To change a single value in the array:**

1. Copy the array.
2. Change the value in the copied array.
3. Pass the copied array to the `setObjectProperty` method.

Changing the value in the array passed to the `setObjectProperty` method earlier will not work.



# Index

## A

- actions **64**
- aligning **36**

## B

- background
  - editing **19**
- background objects
  - polyline **21**
  - properties **23**
  - shapes **20**
- basic dashboard **49**

## C

- changing
  - layers **25**
- classes
  - main **52**

## D

- dashboard
  - actions **64**
  - adding class symbols **12**
  - basic **49**
  - editing **57**
  - opening **42**
  - reading **67**
  - saving **33**
  - testing **37**
  - writing **67**
- documents
  - multiple **43**
- duplication **57**

## E

- editing **57**
  - polyline **21**
  - properties **23**

## G

- grouping **36**
- GUI
  - components **55, 56**
  - menus **56**
  - toolbars **57**

## I

- IlvDashboardBackgroundBar class **57**
- IlvDashboardAction class **56, 57, 64**
- IlvDashboardContext class **52**
- IlvDashboardDiagram class **51**
  - readBinary method **67**
  - readDashboard method **67**
  - writeBinary method **67**
  - writeDashboard method **67**
- IlvDashboardEditBar class **57**
- IlvDashboardEditor class **52**
- IlvDashboardEditorFrame class **51**
- IlvDashboardMenuBar class **56**
- IlvDashboardPropertyPanel class **60**
- IlvDashboardSymbol class **52**
- IlvDashboardSymbolPalette class **62**
- IlvDashboardTree class **59**
- IlvDiagrammer class **51**
- IlvDiagrammerAction class **64**
- IlvDiagrammerEditBar class **57**
- IlvDiagrammerOverview class **61**
- IlvDiagrammerViewBar class **57**
- IlvGraphic class **52**
- IlvGraphicPaletteBar class **57**

## L

- layers **25**
  - changing **25**
  - lowering **29**
  - raising **29**

- transparency **32**
- visibility **31**

- links

- adding **14**
  - in **16**
  - out **16**
  - points **14**

- loading a palette **40**

## **M**

- multidocument interface **43**

## **O**

- objects

- aligning **36**
  - controlling **57**
  - grouping **36**

- overview **61**

- visibility **61**

## **P**

- palettes **40**

- loading **40**
  - pane **62**

- pane **9**

- overview **61**
  - palettes **62**
  - properties **60**
  - tree **59**

- parameters

- changing **34**

- PATH (UNIX) **9**

- points **22**

- intermediate **14**

- polyline

- points **22**

- properties

- background object **23**
  - editing **23**
  - pane **60**

## **R**

- readBinary method

- IlvDashboardDiagram class **67**

- readDashboard method

- IlvDashboardDiagram class **67**

- running

- unix **9**

## **S**

- Start menu (Windows®) **9**

- symbols

- adding to dashboard **12**

## **T**

- toolbars **57**

- transparency **32**

- tree pane **59**

## **U**

- UML **52**

- UNIX **9**

- URL **44**

## **V**

- visibility **31**

- overview **61**

## **W**

- writeBinary method

- IlvDashboardDiagram class **67**

- writeDashboard method

- IlvDashboardDiagram class **67**

## **Z**

- zoom **61**