



IBM ILOG Views Charts V5.3

User's Manual

June 2009

© **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.

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 information see *<installdir>/license/notices.txt* in the installed product.

Table of Contents

Preface	About This Manual 10 What You Need to Know 10 Manual Organization 10 Notation 11 Typographic Conventions 11 Naming Conventions 11 Related Documentation 12
Part I	Using Charts with IBM ILOG Views Studio 14
Chapter 1	Introducing Charts in IBM ILOG Views Studio 16 Launching IBM ILOG Views Studio with the Charts Extension 16 A Quick Look at the Interface 17 Creating a Chart Object 20 Using the Chart Inspector 20 Chart Inspector Icons 22 General Page 22 Data Sets Page 23 Displayers Page 24

	Projection Page	26
	Scales Page	27
	Layout Page	29
	Miscellaneous Page	31
	Callbacks Page	34
	Using the Chart Legend Inspector	34
Chapter 2	Customizing Charts	38
	Example 1: Charting Temperatures and Pressures of the Week.....	38
	Defining Several Independent Ordinate Scales	39
	Defining a Grid Associated with a Scale	52
	Defining a Related Ordinate Scale	52
	Creating a Stacked or a Side-by-Side Representation	56
	Example 2: Charting Analytic Functions	60
	Using a Data Set Defined by a Script Function	60
	Using Logarithmic Scales	66
	Connecting a Legend to a Chart	68
Chapter 3	Using Polar Charts	72
	Example 1: Representing Values Expressed in Radians.....	73
	Case 1: Applying a Transformation	73
	Case 2: Setting a Starting Angle and a Range	79
	Example 2: Representing Time Values	81
	Creating the Polar Chart	82
	Defining the Data Set	82
	Defining the Displayer	84
	Customizing the Projection	85
	Customizing the Abscissa Scale	85
	Customizing the Ordinate Scale	86
Part II	Using the Charts Library	88
Chapter 4	Introducing the Charts Library	90

	Main Features of the Charts Library	90
	Global Chart Characteristics	90
	Data Features	91
	Graphical Representations of Data	91
	Scale Features	92
	Decorations	92
	Interactors	93
	Feature Illustrations	94
Chapter 5	Chart Basics	96
	What is a Chart?	96
	General Architecture of the Charts Library	99
	Data Classes	100
	Chart Classes	101
	Basic Steps for Creating a Chart	102
	Creating a Simple Cartesian Chart	103
	Creating a Simple Polar Chart	109
	Additional Ways to Customize a Chart	112
	How Charts Work in IBM ILOG Views	115
	Components of a Chart Object	115
	Component Classes of the Charts Library	117
	Using the Component Classes in an IlvChartGraphic Object	120
	How Displayer Objects Draw the Graphical Display	123
Chapter 6	Data Handling	126
	Handling Data Storage	126
	Types of Data Sets	127
	Adding Data Sets to Be Displayed by a Chart	131
	Sharing Data Among Charts	132
	Modifying Data and Updating Charts	134
	Types of Modifications	134
	Updating Charts Automatically	135

	Using Listeners to Catch Data Changes	138
Chapter 7	Chart Layout	144
	Computing the Chart Layout	144
	Setting General Properties of a Chart Layout Object.....	146
	Getting and Setting the Chart Layout Object of a Chart	148
Chapter 8	Data Display	150
	Drawing the Graphical Representations of Data	150
	Using Single Displayers	154
	Scatter Displayer	155
	Polyline Displayer.....	156
	Polygon Displayer	159
	Step Displayer	160
	Stair Displayer	161
	Bar Displayer	162
	3D Bar Displayer	163
	High-Low Displayer	164
	High-Low Bar Displayer	166
	Pie Displayer	167
	Using Composite Displayers	171
	Marked Polyline Displayer	173
	High-Low Open-Close Displayer	175
	Stacked Displayers	177
	Side-by-Side Displayers.....	181
	Adding a Displayer to a Chart	183
	Examples	184
	Customizing Data Display.....	184
	Adding Graphic Information to a Data Point.....	185
	Defining How the Palettes are Applied for the Data Display	190
	Projecting Out-of-Bounds Data Points.....	191
Chapter 9	Scales Display	194

	Drawing the Scales of a Chart	194
	Setting General Properties	195
	Using Single Scale Displayers	197
	Setting General Properties	198
	Predefined Single Scale Displayers	203
	Using Scale Steps Updaters to Compute Scales Graduations	206
	Adding a Scale Displayer in a Chart	208
	Advanced Features for Customizing Scales	210
	Changing the Orientation of the Scales	210
	Defining the Minimum and Maximum Data Values Represented by a Scale	212
	Applying a Transformation to the Data Values Represented by a Scale	213
Chapter 10	Decorations Display	216
	Displaying a Legend	216
	Setting General Properties	218
	Adding a Legend to a Chart	219
	Displaying a Grid	219
	Setting General Properties	220
	Adding a Grid Displayer to a Scale	221
	Displaying a Cursor	224
	Setting General Properties	225
	Adding a Cursor to a Scale	226
Chapter 11	Interacting with Charts	230
	Using the Chart Interactors	230
	Zoom Interactor	232
	Scroll Interactor	232
	Pan Interactor	233
	Crosshair Interactor	233
	Drag-Point Interactor	233
	Highlight-Data-Point Interactor	234
	Information-View Interactor	234

	Select-Data-Points Interactor	235
	Setting an Interactor on a Chart Object	236
	Example	236
Chapter 12	Using Charts to Display Real-Time Data	238
	Automatic Scroll Modes	238
	Using Automatic Scroll Modes to Display Real-Time Data	239
	Scroll Example	240
	Improving Performance When Adding Data Points to a Chart	243
	Releasing the Automatic Update	246
Appendix A	The IivXMLChartData Class	248
	Introducing the IivXMLChartData Class	249
	Tags Definition	250
	data	250
	series	251
	valuesList	251
	valueOperator	251
	property	251
	Customizing Value and Date List Processing	252
Index		254

About This Manual

This *User's Manual* explains how to use the Charts Library of IBM® ILOG® Views and the Charts extension of IBM ILOG Views Studio.

What You Need to Know

This manual assumes that you are familiar with the Microsoft® Windows® or UNIX® environment in which you are going to use IBM ILOG Views, including its particular windowing system. Since IBM ILOG Views is written for C++ developers, the documentation also assumes that you can write C++ code and that you are familiar with your C++ development environment so as to manipulate files and directories, use a text editor, and compile and run C++ programs.

Manual Organization

This manual is divided into two parts that describe how to use the IBM® ILOG® Views Charts Package, and one Appendix.

Part I, *Using Charts with IBM ILOG Views Studio* provides an introduction to the IBM ILOG Views Charts package by presenting several examples of the types of charts and the customizing that can be performed with this package. You will learn how to use the

Charts extension of IBM ILOG Views Studio, which is dedicated to the IBM ILOG Views Charts package. It contains the following chapters:

- ◆ Chapter 1, *Introducing Charts in IBM ILOG Views Studio*
- ◆ Chapter 2, *Customizing Charts*
- ◆ Chapter 3, *Using Polar Charts*

Part II, *Using the Charts Library* gives information on the Charts Library of IBM ILOG Views. It contains the following chapters:

- ◆ Chapter 4, *Introducing the Charts Library*
- ◆ Chapter 5, *Chart Basics*
- ◆ Chapter 6, *Data Handling*
- ◆ Chapter 7, *Chart Layout*
- ◆ Chapter 8, *Data Display*
- ◆ Chapter 9, *Scales Display*
- ◆ Chapter 10, *Decorations Display*
- ◆ Chapter 11, *Interacting with Charts*
- ◆ Chapter 12, *Using Charts to Display Real-Time Data*

Appendix A, *The IlvXMLChartData Class*

Notation

Typographic Conventions

The following typographic conventions apply throughout this manual:

- ◆ Code extracts and file names are written in `courier` typeface.
- ◆ Entries to be made by the user are written in `courier` typeface.
- ◆ Some words appear in *italics* when seen for the first time.

Naming Conventions

Throughout this manual, the following naming conventions apply to the API.

- ◆ The names of types, classes, functions, and macros defined in the IBM® ILOG® Views Charts library begin with `Ilv`.

- ◆ The names of classes as well as global functions are written as concatenated words with each initial letter capitalized.

```
class IlvDrawingView;
```

- ◆ The names of virtual and regular methods begin with a lowercase letter; the names of static methods start with an uppercase letter. For example:

```
virtual IlvClassInfo* getClassInfo() const;  
static IlvClassInfo* ClassInfo() const;
```

Related Documentation

For a description of the IBM® ILOG® Views Charts C++ classes, global functions, type definitions, macros, and error messages, see the online version of the IBM ILOG Views *Charts Reference Manual*.

For information on graphics objects in general, see the IBM ILOG Views *Foundation User Manual*.

For information on using IBM ILOG Views Studio, see the IBM ILOG Views *Studio User's Manual*.

Part I

Using Charts with IBM ILOG Views Studio

This part consists of the following chapters:

- ◆ Chapter 1, *Introducing Charts in IBM ILOG Views Studio* describes the IBM® ILOG® Views Studio with Charts interface.
- ◆ Chapter 2, *Customizing Charts* teaches you how to customize charts by working with Cartesian charts.
- ◆ Chapter 3, *Using Polar Charts* guides you through the process of creating and customizing polar charts.

Introducing Charts in IBM ILOG Views Studio

In this chapter, you will find some basic information to get you started using charts in Studio. You will find information on the following topics:

- ◆ *Launching IBM ILOG Views Studio with the Charts Extension* explains how to access the Charts extension of IBM® ILOG® Views Studio.
- ◆ *A Quick Look at the Interface* briefly describes the Main window and Palettes panel that are displayed at start-up time.
- ◆ *Creating a Chart Object* explains how to create a chart object.
- ◆ *Using the Chart Inspector* shows each notebook page of the Chart inspector and briefly explains what you can do in each of them.
- ◆ *Using the Chart Legend Inspector* shows and describes the inspector that enables you to customize legend objects.

Launching IBM ILOG Views Studio with the Charts Extension

To launch IBM® ILOG® Views Studio with the Charts extension, do the following:

1. Go to the directory `$ILVHOME/studio/<system>` of the IBM® ILOG® Views distribution.
2. Type `ivfstudio -selectPlugins`.
3. A window appears listing the available plug-ins.
4. Check Charts and any other plug-ins you may want to use.

Note: *If you want to construct an application with IBM ILOG Views Studio and, in particular, have access to the Test mode, you also need to check the GUI Application plug-in.*

5. Click OK to validate and launch IBM ILOG Views Studio with the plug-ins you have selected.

A Quick Look at the Interface

When you launch IBM® ILOG® Views Studio with the Charts extension, the Main window with the Palettes panel appears on your screen.

- ◆ The work space on the right of the Main window contains the buffer window(s) created by default. You will use these buffer windows to drag and drop chart objects from the Charts palette. You can use the buffer window on top (the Gadgets buffer window), or create another buffer window by selecting New from the File menu.
- ◆ The Palettes panel on the left displays the palettes of predefined graphic objects available in IBM ILOG Views. The upper pane displays a tree gadget with various items, each corresponding to a particular graphic palette. The lower pane displays the objects contained in the palette selected from the tree.

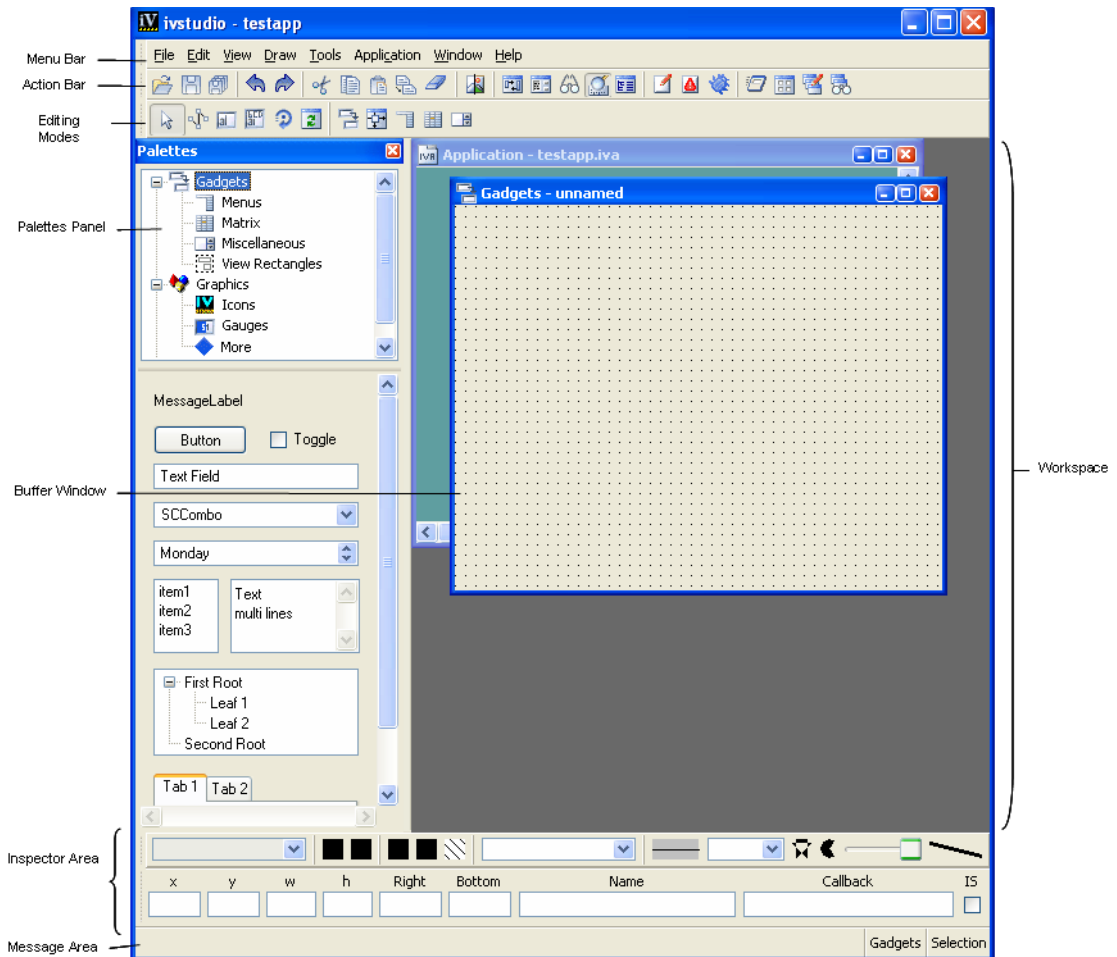


Figure 1.1 Main Window at Start Up

Scrolling down the tree in the upper pane of the Palettes Panel, you see Charts, a subitem of Graphics. When you click Charts, you display the Charts palette in the lower pane.

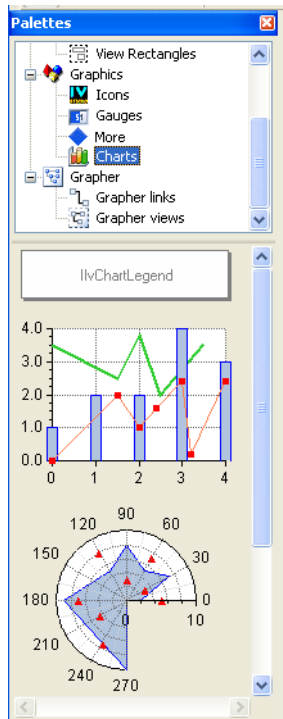


Figure 1.2 Charts Palette

The Charts palette contains the following objects:

- ◆ A chart legend object, which is an instance of the `IlvChartLegend` class. You will use this object to add legends to your charts. The legend object that appears in the palette displays only the name of the class. Legend items will be displayed when the legend object is connected to a chart.
- ◆ Several chart objects, which are all instances of the `IlvChartGraphic` class. These chart objects differ only in the way that they have been customized. Three examples of charts are provided in the palette:
 - **Cartesian chart:** this is a chart object that has been customized to display data expressed in Cartesian coordinates in a standard way.
 - **Polar chart:** this is a chart object that has been customized to display data expressed in polar coordinates in a circular way.
 - **Pie chart:** this is a chart object that has been customized to display a pie.

Creating a Chart Object

To create a chart object, do the following:

1. Click Charts in the upper pane of the Palettes Panel to display the contents of the Charts palette.
2. Select the chart you want from the lower pane of the Palettes panel and drag it to your working buffer window.

Your working buffer window can be the buffer window that is created by default or any other window you have created in the work space. You can create a new buffer window by selecting File > New from the menu bar at the top of the Main window, and then the buffer type you want (2D Graphics, and so on).

When you drag a chart object from the Charts palette to the buffer window, the corresponding `IlvChartGraphic` object surrounded by selection handles appears in the window.

3. Double-click the chart object to open its specific inspector.

The Chart inspector enables you to customize the chart object. The number of parameters you will have to change depends on how different you want your chart to be from the initial chart.

Two dedicated inspectors have been implemented for the IBM® ILOG® Views Charts package:

- ◆ A Chart inspector for chart objects
- ◆ A Chart Legend inspector for chart legend objects

Using the Chart Inspector

You will use the Chart inspector to customize the predefined chart objects to fit your particular needs. The same generic inspector is used for all the chart types: Cartesian, polar, pie, or other.

Once you have launched IBM® ILOG® Views Studio with the Charts Extension and dragged a chart object to your working buffer window, you open the Chart inspector by double-clicking this object. The following window appears:

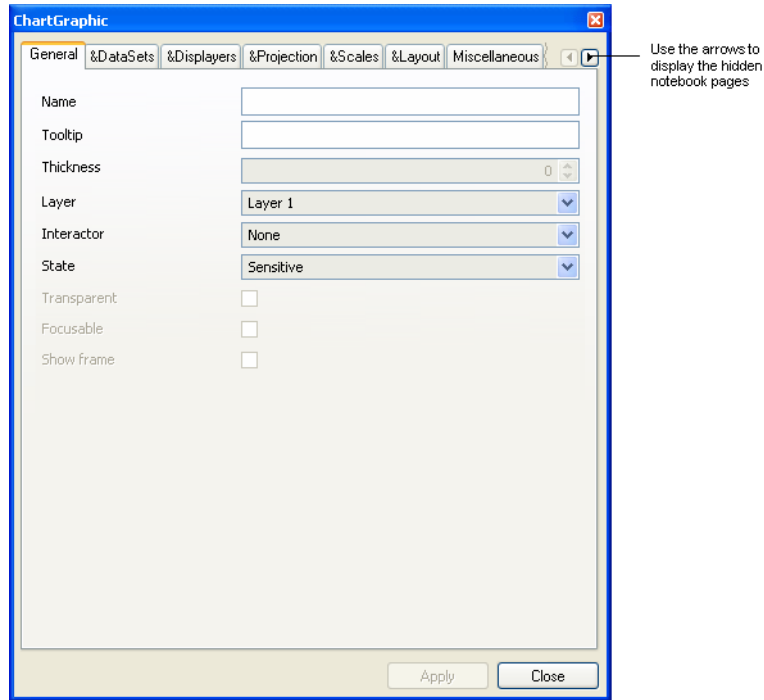


Figure 1.3 *Chart Inspector: General Page*

The Chart inspector contains the following notebook pages:

- ◆ *General Page*
- ◆ *Data Sets Page*
- ◆ *Displayers Page*
- ◆ *Projection Page*
- ◆ *Scales Page*
- ◆ *Layout Page*
- ◆ *Miscellaneous Page*
- ◆ *Callbacks Page*

The first and the last pages (General and Callbacks) apply to all graphic objects in general while the others are specific to the chart objects. You will use these pages to define the various elements of a chart.

To scroll to the hidden pages, use the arrows at the right end of the tab bar.

As you will notice in the illustrations, a number of general-purpose icons are available on some of the Chart inspector pages. These icons are described in the next section and each of the notebook pages is presented in the sections that follow.

Chart Inspector Icons

The name of each icon in the Chart inspector appears as a tooltip when you hold the mouse pointer on the icon.



Add icon Creates a new item *after* the selected item. Depending on the notebook page where you are working, clicking this icon adds a data set in the Data sets list, a row in a data table, a displayer in the Displayers list, an ordinate scale in the Scales list, or a label on the Scales/Steps page when “Labels” is selected in the Step definition list.



Insert icon Creates a new item *before* the selected item. Depending on the notebook page where you are working, clicking this icon inserts a new data set before the selected data set in the Data sets list, a new row before the selected row in a data table, a new displayer before the selected displayer in the Displayers list, a new ordinate scale before the selected ordinate scale in the Scales list, or a new label before the selected label on the Scales/Steps page when “Labels” is selected in the Step definition list.



Clean icon This icon has the same name and effect in all the Chart inspector pages. Clicking this icon clears the contents of the field above. For example, it allows you to clear a data table. All the rows of the data table are erased at one time when you click this icon.



Move up/Move down icons Move the selected item up or down in a list.



Remove icon Erases the selected item. Depending on the notebook page where you are working, clicking this icon erases the selected data set, data row, displayer, ordinate scale, or step label from the corresponding list.

General Page

The General page allows the setting of parameters that are common to all graphic objects.

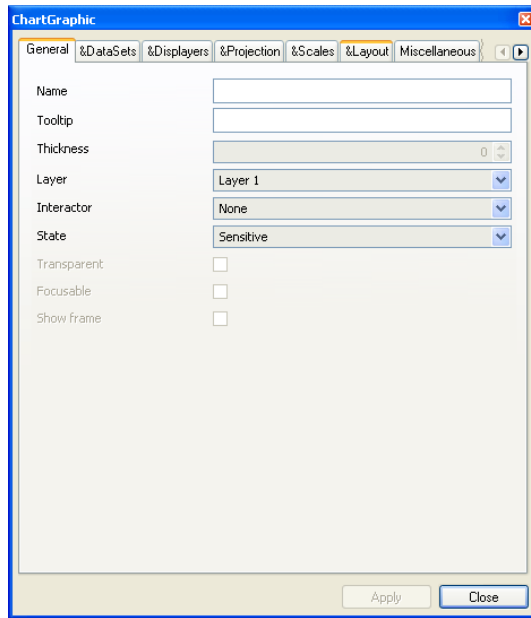


Figure 1.4 Chart Inspector: General Page

Data Sets Page

The Data Sets page lets you define and handle the data sets that will be represented by the current chart.

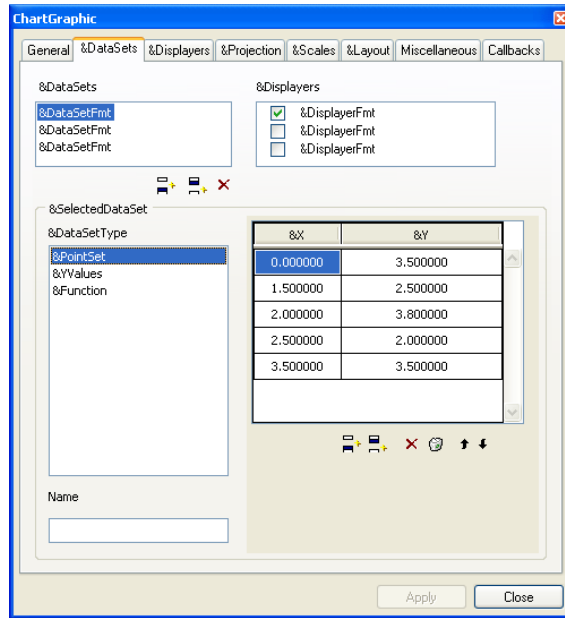


Figure 1.5 Chart Inspector: Data Sets Page

On this page, you can do the following:

- ◆ Add or remove data sets.
- ◆ Select or change the type of a given data set and enter the corresponding data into the data set.
- ◆ Set a name for a given data set.
- ◆ Check or uncheck the displayer(s) that will display a given data set.

To access the information related to a given data set (that is, the type of the data set, the corresponding data, the name and the displayer(s) that will display the data set), the data set must be selected in the Data sets list that shows the defined data sets (top left of the page).

Displayers Page

The Displayers page lets you define and handle the displayers (that is, the graphical representations) that will be used to represent data in the current chart.

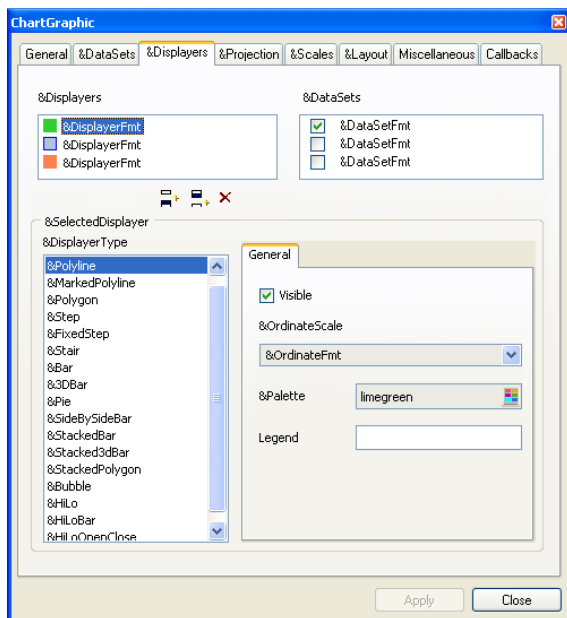


Figure 1.6 Chart Inspector: Displayers Page

On this page, you can do the following:

- ◆ Add or remove displayers.
- ◆ Select or change the type of a given displayer and enter or change its corresponding parameters.
- ◆ Check or uncheck the data set(s) that will be displayed by a given displayer.

To access the information related to a given displayer (that is, the type of the displayer, the corresponding parameters, and the data set(s) that will be displayed by the displayer), the displayer must be selected in the Displayers list that shows the defined displayers (top left of the page).

Note: All the displayer types can be used no matter whether the type of the coordinate system in which the data are expressed is Cartesian or polar (except for the pie displayer that can only be used with a polar coordinate system). The type of the coordinate system appears on the Projection page since the type of the projection that is specified depends on the type of the coordinate system in which the data are expressed.

Projection Page

The Projection page lets you define the type of projection to map the data into screen coordinates.

- ◆ *Cartesian* is used for data expressed in Cartesian coordinates (x, y). The data are represented in a standard manner: the abscissa and ordinate scales representing the x - and y -coordinates, respectively, are orthogonal.

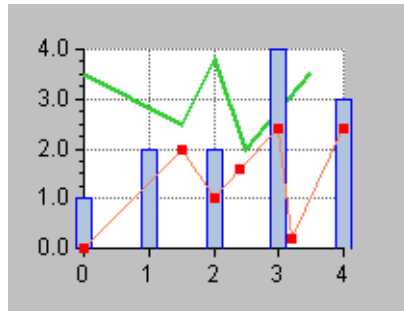


Figure 1.7 Example of a Cartesian Chart

- ◆ *Polar* is used for data expressed in polar coordinates (θ, ρ). The data are represented in a circular way: the θ values are mapped along a circular abscissa scale, while the ρ values are represented along a radial ordinate scale.

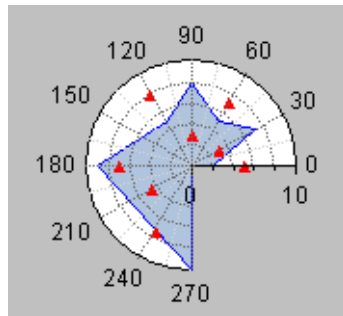


Figure 1.8 Example of a Polar Chart

The Projection page appears as follows:

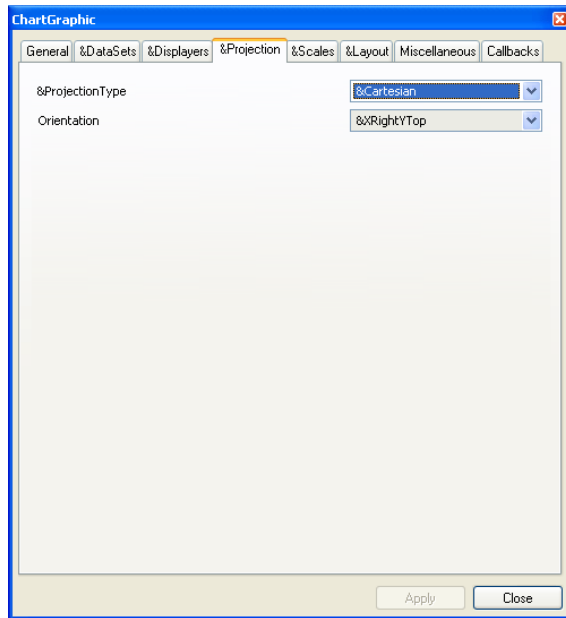


Figure 1.9 *Chart Inspector: Projection Page*

Once you have selected the type of the projection to be applied, you can define:

- ◆ The orientation of the scales for a Cartesian projection.
- ◆ The orientation of the abscissa scale (clockwise or counterclockwise) for a polar projection.

Scales Page

The Scales page lets you define the scales used in the current chart. A chart can have one abscissa scale and as many ordinate scales as you want.

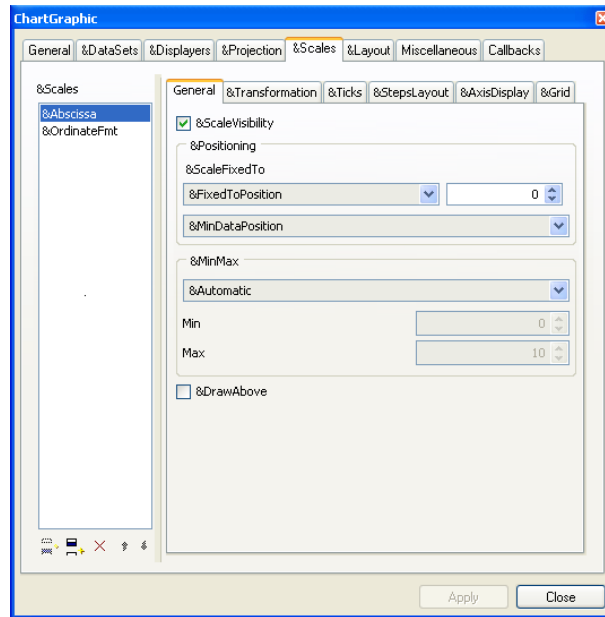


Figure 1.10 Chart Inspector: Scales Page

The Scales page is divided into several subpages that allow you to set various parameters for the selected scale.

- ◆ On the General subpage, you can set:
 - the position of the scale. The scale can be fixed to a data value or to a relative position expressed in pixels.
 - the minimum and maximum data values represented by the scale.
- ◆ On the Transformation subpage, you can set the transformation applied.

The transformations that can be applied are composed of an elementary transformation, optionally followed by a logarithmic transformation. If you want to apply a transformation to the data, you must first select a defined elementary transformation from the drop-down list. Optionally, you can then set a logarithmic transformation. If you want to apply a logarithmic transformation but you do not want to apply a particular elementary transformation before, just select Identity in the drop-down list showing the defined elementary transformations.

- ◆ On the Steps subpage, you specify how the steps of the chart are defined. The steps can be defined automatically or by specifying the number of steps, the step unit, or the labels that will be drawn next to the major ticks.

The term “step” refers to the main graduations of a scale and the term “substep” refers to the secondary graduations of the scale. The term “tick” refers to the marks that are drawn on a scale axis at each step and substep. Major ticks are drawn at each step and minor ticks are drawn at each substep. Labels indicating data values are drawn only for the steps of a scale.

- ◆ On the Ticks display subpage, you specify all the parameters related to displaying ticks. You can specify the size of the ticks, the color, the visibility, the position (outside, inside, or centered on the axis) of the ticks and the step labels, whether step labels are drawn at the axes crossings, and whether overlapping step labels are drawn.
- ◆ On the Axis display subpage, you can specify all the parameters related to displaying an axis. You can specify the visibility, whether an arrow is drawn at the end of the axis, the color of the axis, the label drawn at the end of the axis, and so on.
- ◆ On the Grid subpage, you can specify the parameters for displaying a grid that is associated with the selected scale. You can specify the color of the grid, its visibility, whether only major lines are drawn or whether both major and minor lines are drawn.

Layout Page

The Layout page lets you define the global layout of the chart (that is, the position of the different areas of the chart within its bounding box). There are three areas within the bounding box of the chart.

- ◆ The *drawing area* is the area where the drawing is performed. All the graphical elements that make up a chart (that is, the graphical representations of data, scales, grids and cursors) are drawn within this area. The drawing area is defined by margins relative to the bounding box of the chart.
- ◆ The *data display area* is the area where the data are displayed: no data points can be displayed outside of this area. This area lies inside the drawing area.
- ◆ The *graph area* that represents the extent of all the graphical elements that make up a chart (that is, all the graphical representations of data, scales, grids and cursors of the chart). This area lies inside the drawing area and contains the data display area.

For more details on the chart layout, see Chapter 7, *Chart Layout*.

Figure 1.11 and Figure 1.12 show examples of these areas for a Cartesian chart and a polar chart.

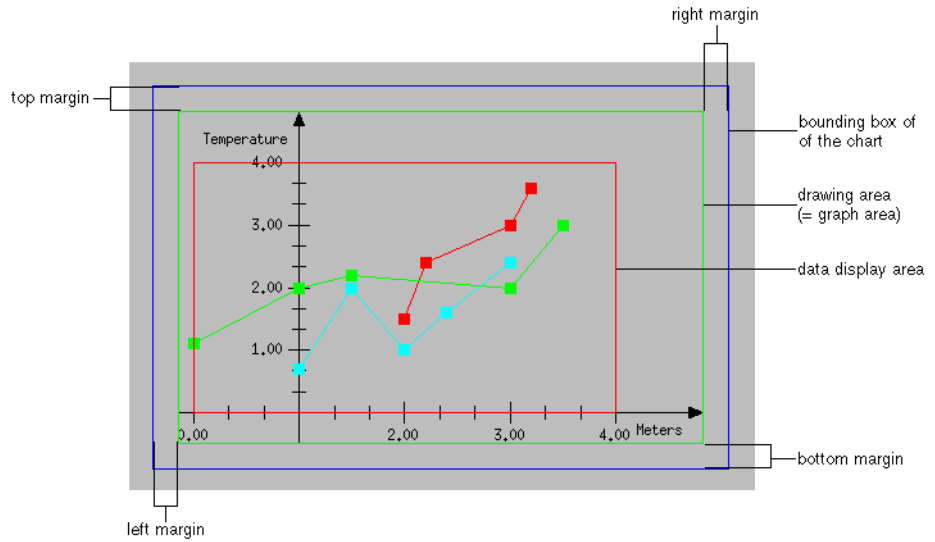


Figure 1.11 Areas within the Bounding Box of a Cartesian Chart

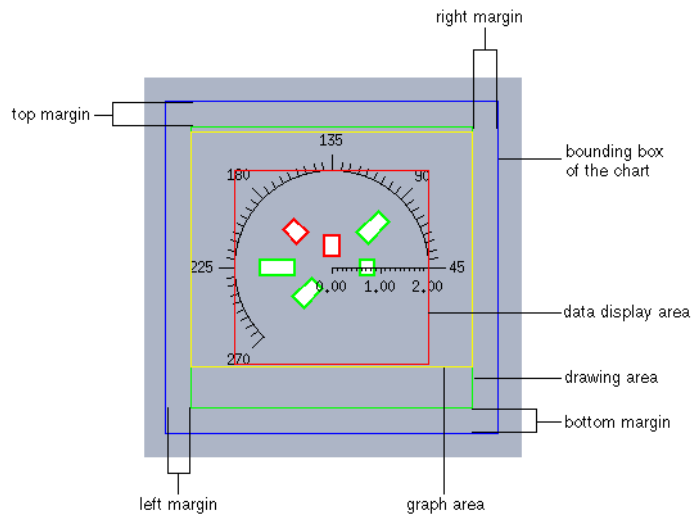


Figure 1.12 Areas within the Bounding Box of a Polar Chart

The Layout page appears as follows:

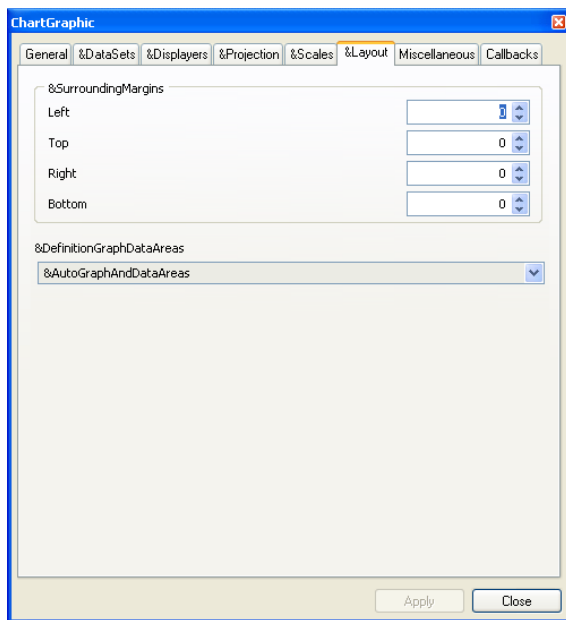


Figure 1.13 Chart Inspector: Layout Page

On this page, you can do the following:

- ◆ Define the drawing area. The drawing area is defined by specifying the margins from the left, right, top, and bottom sides of the bounding box.
- ◆ Define the graph area and the data display area. The graph area and the data display area can be either automatically computed or fixed to a given position.

If “Automatic” is selected, the graph area and the data display area are automatically computed from the drawing area so that the graph area takes up the maximum amount of available space.

If “Fixed graph area” is selected, the graph area is positioned with relative margins from the drawing area and the data display area is automatically computed from the graph area.

If “Fixed data display area” is selected, the data display area is positioned with relative margins from the drawing area and the graph area is automatically computed from the data display area.

Miscellaneous Page

The Miscellaneous page lets you define several parameters that affect the whole chart. The Miscellaneous page appears as follows:

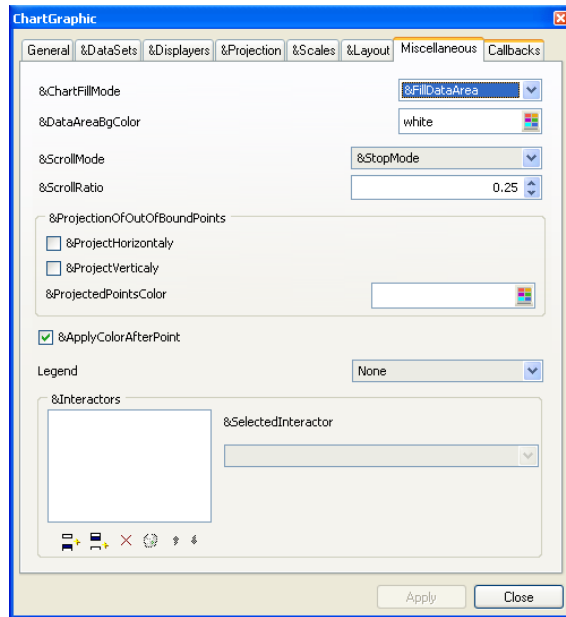


Figure 1.14 Chart Inspector: Miscellaneous Page

On this page, you can do the following:

- ◆ Set the filling mode of the chart and the color used to fill the data display area.

Note: If the filling mode is set to “Filled data and graph areas”, the graph area will be filled with the background color of the chart palette.

- ◆ Set scroll parameters: the scroll mode, and the scroll ratio. These parameter define the behavior of a chart when new data to display are added “on the fly.”

Three scroll modes are available:

- **Stop mode** The new data points that are added are displayed only if they belong to the current data display area.
- **Shift mode** When new data points that do not belong to the current data display area are added, the chart shifts towards the minimum values in order to have enough space to display the new data points.
- **Cyclic mode** When new data points that do not belong to the current data display area are added, the chart scrolls in a cyclic way. The display is not shifted as in the shift scroll mode, but the new data points are simply drawn at the location of the minimum values, thus erasing them.

- ◆ Specify how out-of-bounds data points should be projected on the chart. Out-of-bounds data points are data points that fall outside the area where the data are displayed.

If you specify a vertical or horizontal projection, the out-of-bounds data points are projected on the limits of the data display area. This feature is available for continuous data representations, such as polylines. The effect of the projection is to fill the parts of the polyline that are not drawn because they are outside the data display area. You can also associate a specific palette to represent the projected data points.

- ◆ Specify how the palette that is defined for a given data point is applied: before or after the data point.


The IBM ILOG Views Charts package allows you to define a specific palette that will be applied to a given data point. This palette is independent of the palette that is applied to the whole graphical representation of the data set to which the data point belongs. This specific palette can only be defined by code. (See *Adding Graphic Information to a Data Point* on page 185 for more details.)

The parameter specifying whether the specific palette is applied before or after the data point can be used only for those displayers for which there is a continuous graphical representation linking several data points (such as polyline displayers, step displayers, and so on). For displayers for which there is a graphical representation by data point (such as scatter displayers, bar displayers, and so on), the specific palette is simply applied to the graphical representation of the data point.

- ◆ Specify the legend that is connected to the chart.
- ◆ Specify the interactors defined for the chart.

Several interactors can be set on the same chart. The different interactors that are set are considered in the order they appear in the list showing the interactors set for the current chart. The order in which the interactors appear in the list indicate the priority for event dispatching.

Notes:

1. *For the defined interactors to work, the Chart interactor must be set on the General Page of the Chart inspector.*
2. *You must be in Test mode, and for this the GUI Application plug-in must have been loaded. To go into Test mode, click the Test icon  in the Action toolbar of the Main window. To leave Test mode, just close the Test window that was opened when you entered Test mode.*
3. *The ChartInfoView interactor works only if the buffer that contains the chart is a Gadgets or Prototype Instances (Gadgets) buffer. To see the types of buffers that can be created, choose New from the File menu at the top of the Main window.*

Callbacks Page

The Callbacks page lets you set the callbacks for the chart objects. The Callbacks page appears as follows:

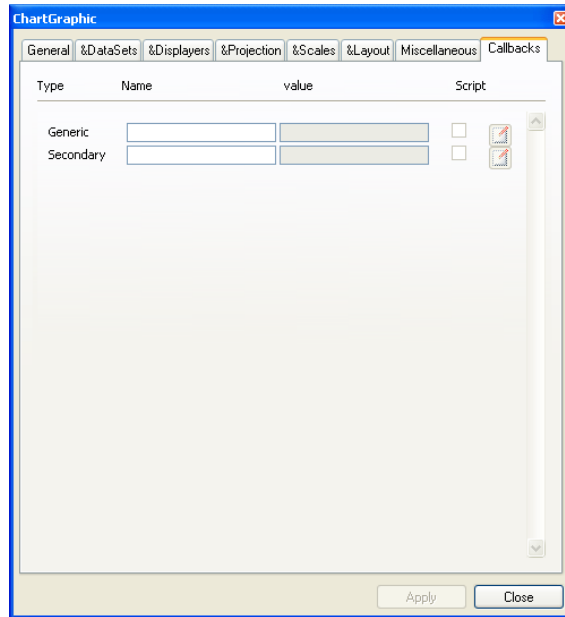


Figure 1.15 Chart Inspector: Callbacks Page

The Callbacks page allows you to set the following:

- ◆ **Name:** The function name of the callback.
- ◆ **Value:** The value for the callback.
- ◆ **Script:** Check this box if you want to use IBM ILOG Script. The button to the right becomes active when the box is selected. Clicking the button show you the callback source code in the Script Editor of the Main window.

Using the Chart Legend Inspector

You will use the Chart Legend Inspector to customize the legend you add to your chart.

Once you have launched IBM® ILOG® Views Studio with the Charts extension and dragged a chart legend object to your working buffer, you can open the Chart Legend inspector by double-clicking this object. Clicking the Specific tab displays the following page:

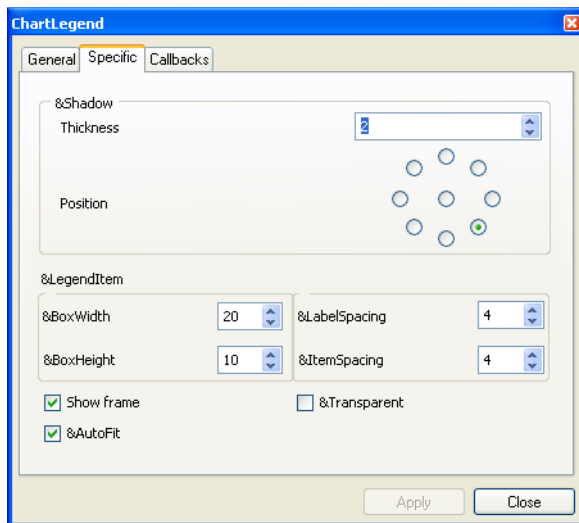


Figure 1.16 Chart Legend Inspector: Specific Page

The Chart Legend Inspector contains the following notebook pages:

- ◆ General
- ◆ Specific
- ◆ Callbacks

The General and Callbacks pages pertain to graphic objects in general while the Specific page contains specific settings for the chart legend object. You will use the Specific page to customize the layout of a given chart legend.

On the Specific page, you can set the following parameters:

- ◆ **Shadow thickness and Shadow position:** the thickness and position of the shadow of the frame surrounding the legend.
- ◆ Dimensions and spacing for the graphical part of the legend items:
 - **Width of graphic part and Height of graphic part:** the width and height of the area where the graphical part of the legend items are drawn. The graphical part of a given legend item is a small drawing whose shape depends on the associated graphical representation of data.
 - **Label Spacing:** the space between the area where the graphical part is displayed and the text of a legend item.
 - **Item Spacing:** the space between two legend items.
- ◆ **Show frame:** specifies whether the frame surrounding the legend appears or not.

- ◆ **Transparent:** specifies whether the legend is transparent or not. A legend is transparent when it is displayed without a background and a shadow.
- ◆ **Automatically fit to contents:** specifies if the legend is automatically resized to fit the legend items it contains.

Customizing Charts

In this chapter, you will be working with Cartesian charts to learn how to customize charts using IBM® ILOG® Views Studio. You will see two examples.

- ◆ In *Example 1: Charting Temperatures and Pressures of the Week*, you will see how to:
 - Use several independent ordinate scales.
 - Define a grid associated with a scale.
 - Define a related ordinate scale.
 - Create a stacked or a side-by-side representation.
- ◆ In *Example 2: Charting Analytic Functions*, you will see how to:
 - Use a data set defined by a script function.
 - Use logarithmic scales.
 - Connect a legend to a chart.

Example 1: Charting Temperatures and Pressures of the Week

The data you are going to represent in your chart are the mean morning and afternoon temperatures, as well as the mean pressure recorded for each day of a week. The

temperatures are expressed in degrees Celsius and the pressures in millibars. These data are listed in the following table:

Table 2.1 *Temperature and Pressure Values*

Day	Morning Temperature (° C)	Afternoon Temperature (° C)	Mean Pressure (millibars)
0	10	16	1012
1	8	12	995
2	12	20	1015
3	15	25	1020
4	14	22	1022
5	14	24	1025
6	13	26	1025

Defining Several Independent Ordinate Scales

Let's assume that you want to represent all the data in Table 2.1 on the same chart. Since the data are expressed in two different units (degrees Celsius and millibars), you need to define two independent ordinate scales, one representing degrees Celsius and the other representing millibars.

Figure 2.1 shows the chart you will be creating. The morning and afternoon temperatures are represented by markers of different colors while the pressures are represented by a marked polyline.

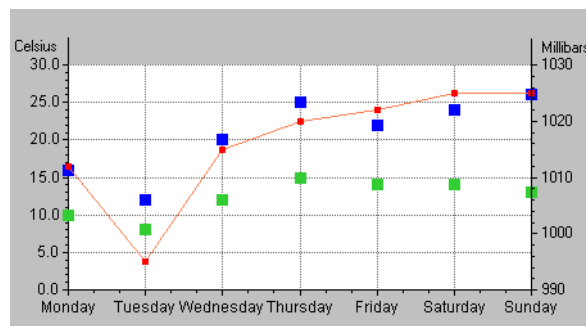


Figure 2.1 *Cartesian Chart with Two Ordinate Scales*

Example 1: Charting Temperatures and Pressures of the Week

You will find the step-by-step instructions to show you how to build this chart. The procedure is broken down into several tasks, which are themselves divided into several steps.

In this part of the example, you will see how to do the following tasks:

- ◆ *Creating a Cartesian Chart*
- ◆ *Defining the Data Sets*
- ◆ *Defining the Displayers*
- ◆ *Customizing the Abscissa Scale*
- ◆ *Customizing the First Ordinate Scale*
- ◆ *Creating and Customizing the Second Ordinate Scale*
- ◆ *Specifying the Ordinate Scale Used by the Displayers*

Creating a Cartesian Chart

Because our example chart is based on a Cartesian chart, the first thing you must do is create the chart.

1. If you have not already done so, launch IBM ILOG Views Studio and display the Charts palette as described in *Launching IBM ILOG Views Studio with the Charts Extension* on page 16.

Note: *It is recommended that you load both Charts and the GUI application plug-ins in order to be able to use the Test mode with Charts.*

2. Drag a Cartesian chart object to your working buffer window.

Your working buffer window can be the buffer window that is created by default or any other window you have created in the work space. You can create a new buffer window by selecting File > New in the menu bar at the top of the Main window and then the buffer type you want (2D Graphics, and so on).

3. Double-click the Cartesian chart object in the buffer window.

The corresponding inspector appears with the General page in front by default.

See *Using the Chart Inspector* on page 20 for an overview of the Chart inspector. This section contains an explanation of each notebook page you may need to use when working with charts.

4. At this stage, it is a good idea to save the buffer containing your chart.


Make sure you select a directory for which you have write access. Also remember to save periodically as you work.


Defining the Data Sets

Because you want to represent morning temperatures, afternoon temperatures, and pressures, you must define three data sets.

Defining the Morning Temperatures Data Set


1. Click the Data sets tab in the Chart inspector to bring this page to the front.

The top of this page shows the list of data sets and the list of displayers created for the current chart. At this stage, you can either use a data set that is already defined for the current chart or create another data set by simply clicking the Insert icon  below the Data sets list.


Just remember to delete the data sets that you will not use since you only need three data sets for our example. To delete a data set, select the data set and click the Remove icon  below the Data sets list.

2. Make sure the first data set is selected in the Data sets list.

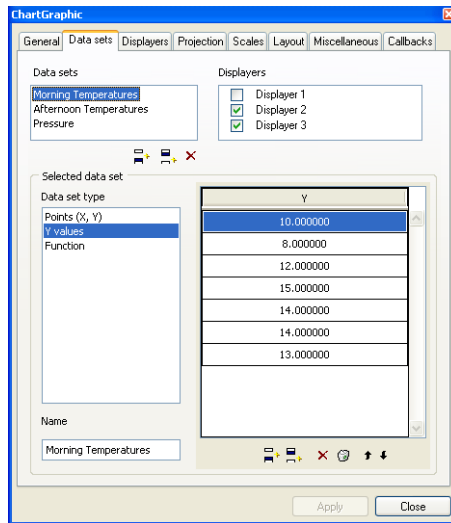
You are going to use this data set to represent the morning temperatures. The days of the week will be displayed along the abscissa scale and the mean temperatures along the ordinate scale. Since the days are referenced by indexes ranging from 0 to 6, you can use a data set of type “Y values” to represent the morning temperatures. By doing this, you only need to enter the temperature values. By default, the values represented on the abscissa scale will be the indexes of the specified Y values.

3. Make sure the selected data set is of type “Y values” in the Data set type list. If not, select “Y values” in the Data set type list.
4. Make sure the Y list in the scrolling list next to the Data set type list is empty. If not, click the Clean icon  below the list to empty it.

You are now going to fill it in with the morning temperatures from Table 2.1.


5. Click the Add icon  below the empty Y list as many times as necessary to create the required number of empty cells.
6. Select the first cell and enter the first temperature value.
7. Repeat step 6 for each cell.
8. In the Name field, change the default name of the data set to `Morning temperatures`.
The Data sets list is updated accordingly.

Example 1: Charting Temperatures and Pressures of the Week



9. Click Apply to validate the change.

Defining the Afternoon Temperatures Data Set


Following the same procedure, you are now going to enter the values for the second data set, the afternoon temperatures. You can either use a data set that is already defined for the current chart or create another data set by simply clicking the Add icon  below the Data sets list once the first data set is selected.

1. Select the second data set in the Data sets list.
2. Make sure this data set is already of type “Y values.” If not, select “Y values” in the Data set type list.
3. Repeat steps 4 to 7 in the previous section to enter the afternoon temperatures.
4. In the Name field, change the default name of the data set to *Afternoon temperatures*.

The Data sets list is updated accordingly.

5. Click Apply to validate the change.

Defining the Pressure Data Set

Finally, you are going to enter the values for the third data set that represents the pressures. You can either use a data set that is already defined for the current chart or create another data set by simply clicking the Add icon  below the Data sets list, once the second data set is selected.


1. Select the third data set in the Data sets list.


2. Make sure “Y values” is selected in the Data set type list. If not, select “Y values” in the Data set type list.
 3. Enter the pressure values by repeating steps 4 to 7 of the morning temperatures data set section.
 4. In the Name field, change the default name of the data set to `Pressures`.
The Data sets list is updated accordingly.
 5. Click Apply to validate the change.
- You have now defined all the data sets for the chart.

Defining the Displayers


You are now going to define the corresponding displayers to display the data sets you defined in the previous section.

1. Click the Displayers tab in the Chart inspector to bring this page to the front.

The top of this page shows the list of displayers and the list of data sets created for the current chart. At this stage, you can either use a displayer that is already defined for the current chart or create another displayer by simply clicking the Insert icon  below the Displayers list.

Just remember to delete the displayers that you will not use since you only need three displayers for our example. To delete a displayer, select the displayer and click the Remove icon  below the Displayers list.

2. Make sure the first displayer is selected in the Displayers list.
This displayer will be used to display the `Morning temperatures` data set.
3. For the displayer to display the `Morning temperatures` data set, the toggle in front of the `Morning Temperatures` data set must be checked. If it is not checked, click the toggle.
Next, you must specify the type of displayer for the data set. Because you want to represent the morning temperatures with markers, you must specify a scatter displayer.
4. Make sure “Scatter” is selected in the Displayer type list. If it is not selected, select “Scatter.”
5. Click Apply to validate the changes.

Following the same procedure, you are now going to define the displayer for the `Afternoon Temperatures` data set. You can either use a displayer that is already defined for the current chart or create another displayer by simply clicking the Add icon  below the Displayers list once the first displayer is selected.


1. Select the second displayer from the Displayers list.

Example 1: Charting Temperatures and Pressures of the Week

2. For the displayer to display the `Afternoon temperatures` data set, the toggle in front of the `Afternoon Temperatures` data set must be checked. If it is not checked, click the toggle.

Next, you must specify the type of displayer for the data set. Because you want to represent the afternoon temperatures with markers, you must specify a scatter displayer.

3. Make sure “Scatter” is selected in the Displayer type list. If it is not selected, select “Scatter.”
4. Click Apply to validate the changes.

Following the same procedure, you are also going to define the displayer displaying the `Pressures` data set. You can either use a displayer that is already defined for the current chart or create another displayer by simply clicking the Add icon  below the Displayers list once the second displayer is selected.

1. Select the third displayer from the Displayers list.
2. For the displayer to display the `Pressures` data set, the toggle in front of the `Pressures` data set must be checked. If not, check this toggle by clicking it.

Next, you must specify the type of displayer for the data set. Because you want to represent the pressures with a marked polyline, you must specify a marked polyline displayer.
3. Make sure “Marked polyline” is selected in the Displayer type list. If not, select “Marked polyline” in the Displayer type list.
4. Click Apply to validate the changes.

You have now defined the displayers for the three data sets to be displayed in your chart.

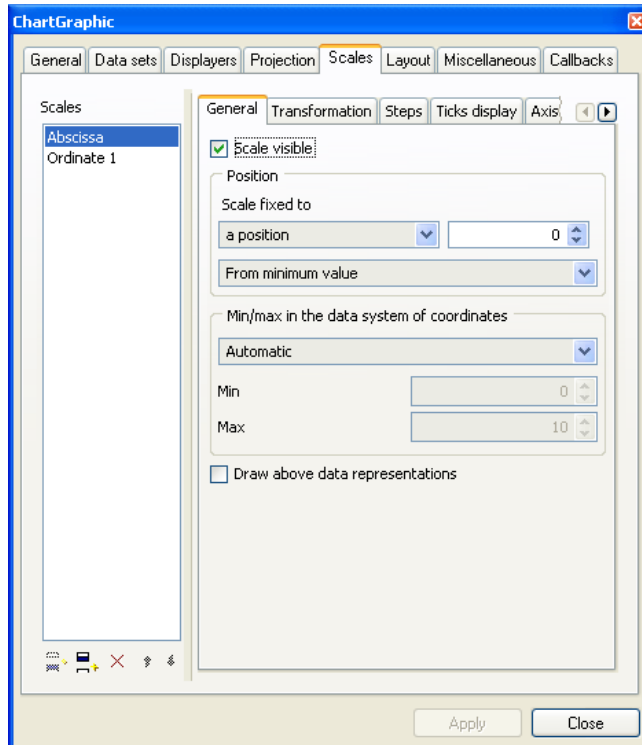
Customizing the Abscissa Scale

For this example, you want to define labels that will be displayed next to the steps of the scale instead of the indexes of the days. To customize the abscissa scale in this way, do the following:

1. Click the Scales tab in the Chart inspector to bring this page to the front.
2. Make sure Abscissa is selected in the Scales list.

The right side of the Scales page is divided into several subpages that describe the properties of the selected scale.

3. Display the General page of the Chart Inspector and make sure that the minimum and maximum data values represented on the current scale are automatically computed. If not, select “Automatic” in the Min/Max drop-down list.




4. Click the Steps tab to bring this page to the front.

By default, the steps are labeled with floating values that correspond to the data. In this example, they are labeled with the indexes of the days (from 0 to 6). However, for better understanding, we prefer to display the names of the days.

5. Choose Labels in the Step definition list.

A rectangular area and the “Selected label” text field appear below this drop-down list. The rectangular area is empty because you have not created any labels yet.

6. To create a label, click the “Add a label” icon  below the empty rectangular area.

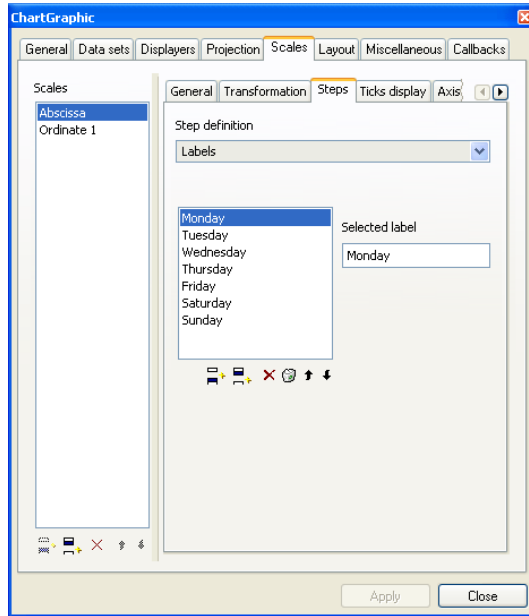
A selected X appears at the top of the rectangular area and the string `&DefaultStepLabel` appears in the “Selected label” field.

7. Select `&DefaultStepLabel` and type the name of the first day of the week, Monday.

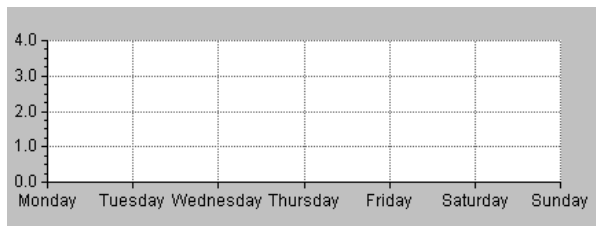
The label you are currently editing is selected in the rectangular area.

8. Repeat steps 6 and 7 to create a label for each day of the week. Make sure the last label you created is selected before you click the “Add a label” icon.

9. Click Apply to validate the new labels.



In the buffer window, the labels on the abscissa scale may overlap. To see each label completely, you may need to resize the chart:



Customizing the First Ordinate Scale

The first ordinate scale of the chart will be graduated in degrees Celsius. To customize the scale in this way, you need to perform the following tasks:

- ◆ *Setting the Minimum and Maximum Values*
- ◆ *Setting the Scale Graduations*
- ◆ *Labeling the Axis*

The Scales page should already be selected in the Charts inspector, so you can continue with these tasks.

Setting the Minimum and Maximum Values

1. Select Ordinate 1 in the Scales list.

The notebook pages on the right side of the page describe the properties of the selected scale.

2. Make sure the General tab is displayed in front.

The scale must be attached to a position that is defined with respect to the minimum data value. If not, select “a position” from the “Scale fixed to” drop-down list, set 0 in the position box, and select “From minimum value” in the drop-down list that appears below. These settings are used to attach Ordinate 1 to the position of the minimum data value represented on the abscissa scale.

The scale Ordinate 1 will be used to represent the temperatures in degrees Celsius. The minimum and maximum data values of the scale must therefore include at least all the temperatures listed in Table 2.1 on page 39. Otherwise, some temperatures will not be displayed.

3. On the Scales/General notebook page, make sure “User-defined” is selected from the Min/max drop-down list. Then enter 0 in the Min field and 30 in the Max field.
4. Click Apply to validate the changes.

Setting the Scale Graduations

1. Click the Steps tab to bring this page to the front.

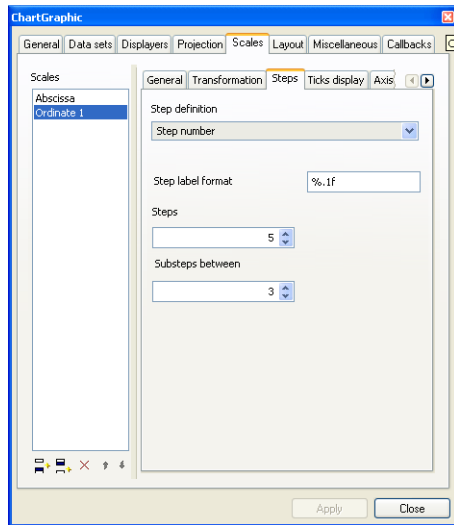
For scale Ordinate 1, you want to define the steps by specifying a step unit.

2. Make sure “Step unit” is selected in the Steps definition list.
3. Enter 5 in the “Step unit” field.

This means that the scale will display one major tick for every five degrees Celsius. (You can either type the value or use the down and up arrows to select the value).

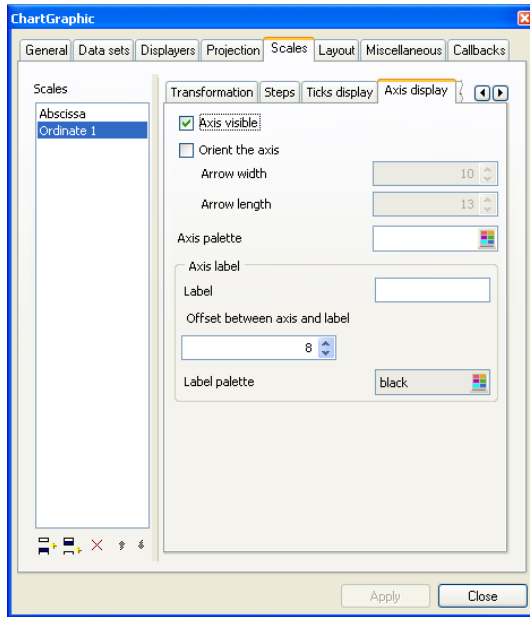
4. Enter 4 as the number of substeps between two steps.
5. Click Apply to validate.

Example 1: Charting Temperatures and Pressures of the Week



Labeling the Axis

1. Click the Axis display tab to bring this page to the front.
2. In the Axis label box, type Celsius in the Label text field.
3. Click Apply to validate the change.




Creating and Customizing the Second Ordinate Scale

Because it will be used to display the pressure values, the second ordinate scale must be graduated in millibars. Unlike the first ordinate scale, which already existed by default, the second one needs to be created before you can customize it.

The Scales page should already be selected in the Charts inspector, so you can continue with these tasks.

Creating Ordinate 2 Scale

1. Select the last created scale (Ordinate 1 at this stage) in the Scales list and click the “Add an ordinate scale” icon  below the Scales list.

A new scale, named Ordinate 2, is added to the list and highlighted. The notebook pages on the right describe the properties of the selected scale.

2. Click the General tab to bring this page to the front.

Customizing Ordinate 2 Scale

To customize the Ordinate 2 scale, you need to perform the following tasks:

- ◆ *Setting the Position and Min/Max Values of the Scale*
- ◆ *Setting the Scale Graduations*
- ◆ *Indicating that Labels Must Be Drawn at the Axes Crossings*

◆ *Labeling the Axis*

Setting the Position and Min/Max Values of the Scale

The General page should be displayed on the Scales page. On the General page, you will specify the position on the abscissa scale to which Ordinate 2 is attached, as well as the minimum and maximum values that will be represented by the scale Ordinate 2. The new scale is attached by default to the position of the minimum data value represented on the abscissa scale (as is Ordinate 1). The minimum and maximum values are computed automatically by default. Therefore, your first task is to change the scale positioning and the way the minimum and maximum values are defined.

1. In the Position box, select 0 as the position.
2. Select “From maximum value” in the drop-down list below.

This specifies that the position of the second ordinate scale is relative to the position of the maximum data value represented on the abscissa scale.

3. Select “User-defined” in the Min/max drop-down list so that you can specify the minimum and maximum values that will be represented by the scale.

Ordinate 2 will represent pressures in millibars. Therefore, the minimum and maximum data values used for the scale must include at least all the pressures given in Table 2.1 on page 39. Otherwise, some pressures will not be displayed.

4. Type 990 in the Min field and 1030 in the Max field.
5. Click Apply to validate the changes.

Setting the Scale Graduations

1. Click the Steps tab to bring this page to the front.
2. Make sure the “Step unit” option is selected.
3. Enter 10 as the step unit.

This means that there will be one major tick displayed for every ten millibars.

4. Enter 4 as the number of substeps between two steps.
5. Click Apply to validate.

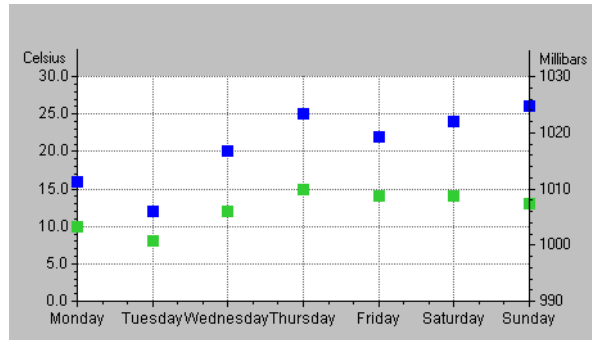
Indicating that Labels Must Be Drawn at the Axes Crossings

1. Click the Ticks display tab to bring this page to the front.
2. Click the “Draw labels on axes crossings” toggle button.
3. Click Apply to validate.

Labeling the Axis

1. Click the Axis display tab to bring this page to the front.
2. In the Axis label box, type `Millibars` in the Label text field.
3. Click Apply to validate.

You have now defined the second ordinate scale. At this stage, your chart should look like this:



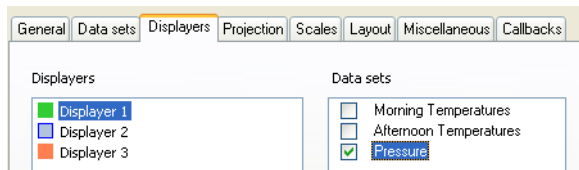
Specifying the Ordinate Scale Used by the Displayers

You must specify the ordinate scale that is used by each displayer to display the data. This is done on the General page of each displayer.

Currently, the default setting (Ordinate 1) is true for the `Morning temperatures` and `Afternoon temperatures` data sets, but false for the `Pressures` data set. The `Pressures` data set must be represented on Ordinate 2 since this scale was created for this purpose.

1. Click the Displayers tab of the Chart inspector to bring this page to the front.
2. Select Displayer 3 in the Displayers list.

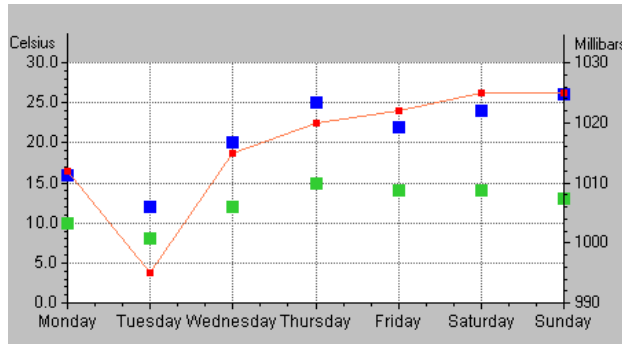
This displayer is associated with the `Pressures` data set, as shown by the checked toggle button in the Data sets box.



3. On the General subpage, select Ordinate 2 from the “Ordinate scale” drop-down list.
4. Click Apply to validate.

Example 1: Charting Temperatures and Pressures of the Week

The chart you obtain at this stage should be similar to Figure 2.1 on page 39.



Defining a Grid Associated with a Scale

To customize our chart further, we can define a grid and associate it with a given scale.

1. Click the Scales tab in the Chart inspector to bring this page to the front.
2. Select the scale with which you want to associate a grid in the Scales list.
3. Click the Grid tab to bring this page to the front.
4. Check the toggle “Use a grid” by clicking it to create a grid and associate it with the selected scale.

Once the toggle is checked, the frame “Grid display” appears that allows you to customize the created grid.

Defining a Related Ordinate Scale

Let’s assume that now you want to add another ordinate scale to represent the temperatures so that you can show the correspondence between degrees Celsius and degrees Fahrenheit. This second temperatures scale will represent the same data as Ordinate 1, but will be graduated in degrees Fahrenheit instead of degrees Celsius.

Figure 2.2 shows the chart you will obtain when you add the Fahrenheit scale:

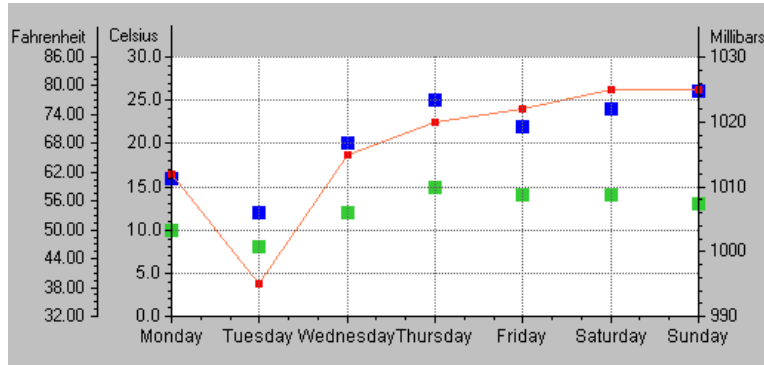


Figure 2.2 *Defining a Related Ordinate Scale*

In this part of the example, you will see how to define a transformation to be applied to the data values represented by a scale.

To define this related ordinate scale, you need to perform the following tasks:

- ◆ *Creating Ordinate 3 Scale*
- ◆ *Customizing Ordinate 3 Scale*

Creating Ordinate 3 Scale

To create the Fahrenheit scale, follow the same procedure as you did to create the second ordinate scale (see *Creating Ordinate 2 Scale* on page 49).

Customizing Ordinate 3 Scale

To customize the Ordinate 3 scale, you need to perform the following tasks:

- ◆ *Setting the Position and Min/Max Values of the Scale*
- ◆ *Applying a Transformation*
- ◆ *Setting the Scale Graduations*
- ◆ *Indicating that the Labels Must Be Drawn at the Axes Crossings*
- ◆ *Labeling the Axis*

Setting the Position and Min/Max Values of the Scale

On the General subpage of the Scales page, you are going to specify the position to which Ordinate 3 is attached, as well as the minimum and maximum values that will be represented on this scale.

1. In the “Scale fixed to” drop-down list, make sure that “a position” is selected. Then select or type -60 in the field next to this option.

Example 1: Charting Temperatures and Pressures of the Week

This position is specified relative to the position of the minimum data value represented on the abscissa scale as indicated by the option “From minimum value.” Make sure this option is selected.

By default, the minimum and maximum values are computed automatically. You must therefore change the way the minimum and maximum values are defined before you can set these values.

2. Select “User-defined” in the Min/max drop-down list.

Ordinate 3 will represent the temperatures in degrees Fahrenheit. The minimum and maximum values must be the same as those set for Ordinate 1 since Ordinate 3 will represent the same data values as Ordinate 1. Ordinate 1 displays graduations in degrees Celsius while Ordinate 3 will display the corresponding values in degrees Fahrenheit.

3. Enter 0 in the Min field and 30 in the Max field, just as you did for scale Ordinate 1.
4. Click Apply to validate the changes.

Applying a Transformation

We want to represent the same data values on scale Ordinate 3 as we represented on scale Ordinate 1. However, on scale Ordinate 3, the values should be expressed in degrees Fahrenheit rather than degrees Celsius. To do this, we have to specify a transformation from degrees Celsius to degrees Fahrenheit for Ordinate Scale 3. Follow these steps:

1. Click the Transformation tab on the Scales page to bring this page to the front.

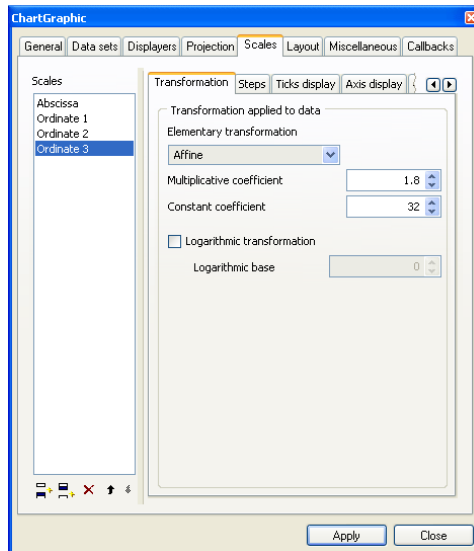
This is where you will set the transformation that will convert degrees Celsius to degrees Fahrenheit.

Converting degrees Celsius to degrees Fahrenheit is done by the following formula:

$$F = \frac{9}{5} \times C + 32$$

where C is a data value expressed in degrees Celsius and F is the equivalent temperature in degrees Fahrenheit.

2. Select Affine from the Elementary transformation list.
3. Enter 1.8 in the Multiplicative coefficient field and 32 in the Constant coefficient field.
4. Click Apply to validate the changes.



Setting the Scale Graduations

1. Select the Steps tab.
2. Make sure “Step number” is selected in the Step definition list.
3. Set 10 as the number of steps and 3 as the number of substeps between steps.
4. Click Apply to validate the changes.

Indicating that the Labels Must Be Drawn at the Axes Crossings

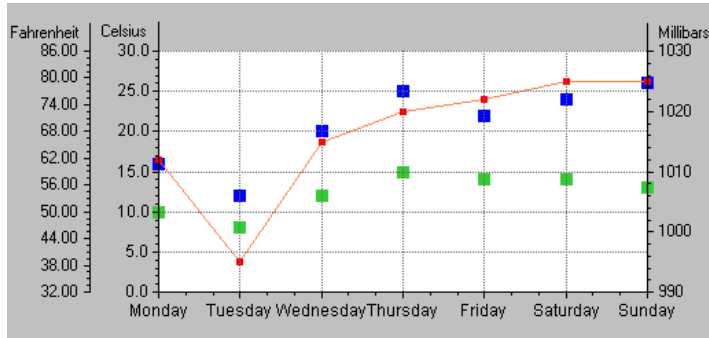
1. Click the Ticks display tab to bring this page to the front.
2. Check the toggle “Draw labels on axes crossings” by clicking it.
3. Click Apply to validate the changes.

Labeling the Axis

1. Select the Axis display page.
2. In the Axis label box, type Fahrenheit in the Label text field.
3. Click Apply to validate the changes.

At this point, your chart should look like the following figure. Now, you can read a given temperature value either in degrees Celsius or in degrees Fahrenheit.

Example 1: Charting Temperatures and Pressures of the Week



Creating a Stacked or a Side-by-Side Representation

This section explains how to create a stacked representation. The same procedure can be used to create a side-by-side representation.

We will continue to use our example chart, which shows the temperatures and pressures (see Table 2.1 on page 39). To better highlight the day of the week with the highest mean temperature, you are going to stack the morning and afternoon temperatures.

Figure 2.3 shows how your chart will appear.

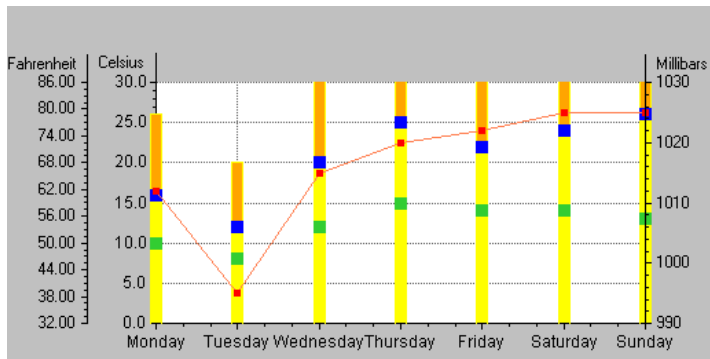



Figure 2.3 Final Stacked-Bar Chart

To create the stacked-bar representation, you will perform the following tasks:

- ◆ *Creating the Stacked-Bar Displayer*
- ◆ *Changing the Color of the Bars*
- ◆ *Changing the Maximum Value of the Ordinate Scales*

Creating the Stacked-Bar Displayer

Starting from the chart you obtain once you have created and customized the third ordinate scale (see the previous section *Defining a Related Ordinate Scale* on page 52), follow the steps below to create a stacked-bar displayer.

1. Select the Displayers page of the Chart inspector.
2. Make sure Displayer 1 is selected.
3. Click the “Insert a displayer before” icon  below the Displayers list.

A new displayer called “Displayer 1” is inserted before the others. (Displayers are displayed in the order in which they are listed.) The new displayer is automatically highlighted.

4. Select Stacked bar from the Displayer type list.

In the Displayers list, an indented tree item also named Displayer 1 is added under Displayer 1.

Notice that at this stage, the stacked-bar displayer Displayer 1 only represents the *Morning* temperatures data set, as indicated by the state of the toggle button in the Data sets box. You want Displayer 1 to display the *Afternoon* temperatures data set as well.

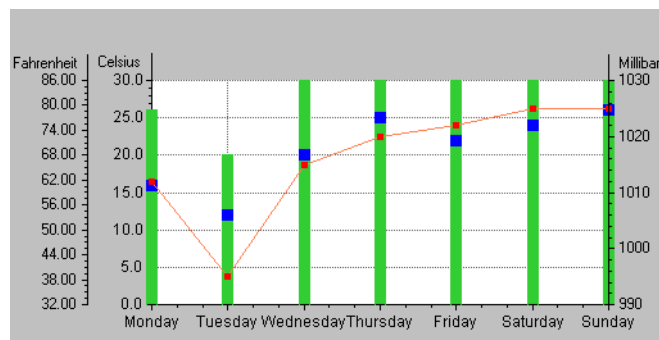
5. In the Data sets list, check the toggle box in front of the *Afternoon* temperatures data set.

Now the stacked-bar displayer Displayer 1 will display both the *Morning* temperatures and *Afternoon* temperatures data sets.

This creates the Displayer 2 subitem.

6. Click Apply to validate the changes.

The chart now looks like this.



Example 1: Charting Temperatures and Pressures of the Week

Seemingly, at this stage, only one bar is visible. However, if you look at the Displayers list on the Displayers page, you can see that the root Displayer 1 is composed of two child displayers that are associated with the morning temperatures and the afternoon temperatures, respectively. These two displayers have the same foreground and background colors. This is why their graphical representations seem to merge into one single bar. To differentiate the morning from the afternoon temperatures within the stacked bars, you must change the colors.

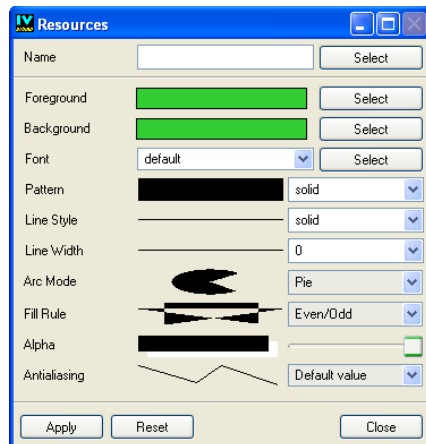
Changing the Color of the Bars

1. In the Displayers list, select the subitem Displayer 1 under the main Displayer 1.

The General page in the right lower corner of the Displayers page shows information on the selected displayer.

2. At the right end of the Palette field, click the small icon.

The Resources panel showing the settings of the palette associated with the displayer opens.

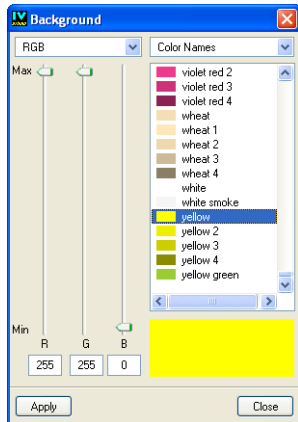


You will keep the default foreground color used to draw the borders of the bars, but you are going to change the background color used to fill the bars.

3. In the Resources panel, click the Select button next to the Background box.

The Background panel that opens lists the available colors.

4. Scroll down the color list to select Yellow. Click Apply in the Background panel to validate this change. Then click Close to close this window.



5. Click Apply in the Resources panel and then Close to close the Resources panel.
6. In the Chart inspector, click Apply to validate this change.
7. To change the color of the second displayer, select the subitem Displayer 2 of the main Displayer 1 in the Displayers list. Repeat steps 2 to 6 to set the background color to Orange.

At this stage, you can see that the stacked bars appear in two different colors. However, some of the bars are truncated. This is because the sum of the morning and afternoon temperatures is sometimes greater than 30, the upper limit set to the scale Ordinate 1. In order to see all of the bars that are truncated, you will need to change this upper value.

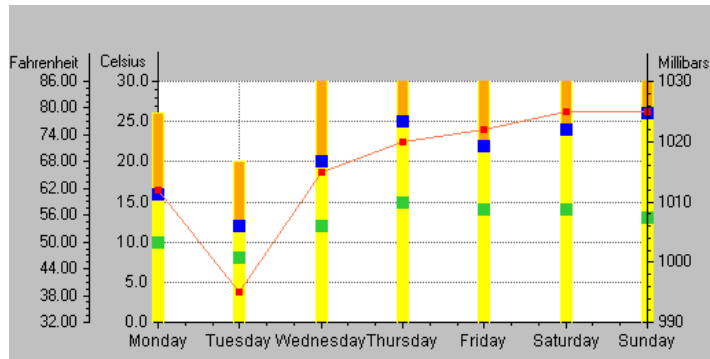
Changing the Maximum Value of the Ordinate Scales

1. In the Chart inspector, select the Scales page.
2. Select Ordinate 1 in the Scales list.
3. On the General subpage, change the value in the Max field from 30 to 40, which is the maximum sum of the morning and afternoon temperatures in our example.
4. Click Apply to validate the change.

Now that you have changed the maximum value of Ordinate 1, you must also change the maximum value represented on Ordinate 3, which converts degrees Celsius to degrees Fahrenheit. Otherwise, the two scales will no longer match.

5. To change the maximum value of Ordinate 3, select Ordinate 3 in the Scales list. Repeat steps 3 and 4.

Your chart should now look like this:



You have now completed the Cartesian chart example representing temperatures (cumulated or not) both on a Celsius scale and on a Fahrenheit scale, as well as pressures on a millibars scale. In working through this example, you have learned how to do the following:

- ◆ Define data sets and their displayers.
- ◆ Create and customize scales.
- ◆ Create and customize a grid associated with a scale.
- ◆ Define a transformation to be applied to the data values represented by a scale.
- ◆ Create a stacked representation.

You can now go on to the second example. In the second example, you are going to represent analytic functions in a chart. In addition to learning more about how to define data sets and displayers and customize scales, you will also learn how to use logarithmic scales and how to connect a legend to a chart.

Example 2: Charting Analytic Functions

The example in this section is based on charting analytic functions of type $y = f(x)$. This section is divided as follows:

- ◆ *Using a Data Set Defined by a Script Function*
- ◆ *Using Logarithmic Scales*
- ◆ *Connecting a Legend to a Chart*

Using a Data Set Defined by a Script Function

Let's suppose you want to represent both the square and the square root functions on the same chart. To do this, you must define specific data sets.

Figure 2.4 shows a chart representing these curves. The two functions are represented by different-colored polylines.

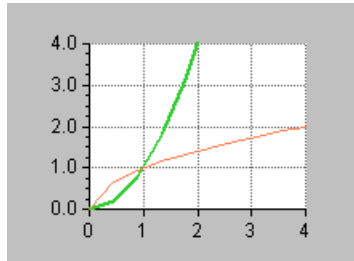


Figure 2.4 Charting the Square and Square Root Functions

This section contains the step-by-step instructions to show you how to build this chart. The procedure is broken down into several tasks, which are themselves divided into several steps.

In this part of the example, you will do the following tasks:

- ◆ *Creating a Cartesian Chart*
- ◆ *Defining the Data Sets*
- ◆ *Defining the Displayers*

If you are not sure how to launch the Chart Studio or how to create a chart, read sections *Launching IBM ILOG Views Studio with the Charts Extension* on page 16 and *Creating a Chart Object* on page 20.

Creating a Cartesian Chart

Again, the chart in this example is based on a Cartesian chart. The first thing to do is to create the chart object.

1. Drag a Cartesian chart object to your working buffer window.
2. Double-click this object.

The corresponding Chart inspector appears.

See the section *Using the Chart Inspector* on page 20 for an overview of the Chart inspector. This section contains an explanation of each notebook page you may need to use when working with charts.

3. At this stage, it is a good idea to save the buffer containing your chart.


Make sure you select a directory to which you have write access. Also remember to save periodically as you work.


Defining the Data Sets

You must define two data sets: one representing the square function and one representing the square root function.

Defining the Square Function Data Set

1. Click the Data sets tab in the Chart inspector to bring this page to the front.

The top of this page shows the list of data sets and the list of displayers created for the current chart. At this stage, you can either use a data set that is already defined for the current chart or create another data set by simply clicking the Insert icon  below the Data sets list.

Just remember to delete the data sets that you will not use since you only need two data sets for our example. To delete a data set, select the data set and click the Remove icon  below the Data sets list.

2. Make sure the first data set is selected in the Data sets list.

If you want to define a data set using a script function, the type of the data set must be set to the type Function. This data set type has been designed to represent analytic functions of type $y = f(x)$. The function f is represented by a function written in the script language. This function takes a value x as its parameter and returns a computed value y . The number of data points used to represent the corresponding curve is specified by the user. These data points are computed between the interval $[x_{min}, x_{max}]$, also specified by the user.

3. Make sure Function is selected in the Data set type list.

In the right-hand part of the “Selected data set” box, you will see several fields where you are going to enter information to define the selected data set.

4. Enter 10 in the Data count field.


This is the number of data points that will be considered for the data set.

5. Enter 0 in the “X minimum” field and 4 in the “X maximum” field.

6. Type `square` in the Script function field to specify the name of the script function used to define the data set.

7. Type `Square Function` in the Name field to change the data set name.

Before clicking Apply to validate the entries, you must define the script function.

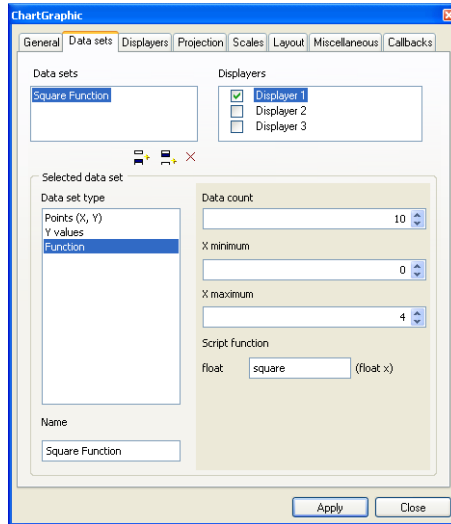
8. Open the Script Editor by clicking the Script Editor icon  in the toolbar of the IBM ILOG Views Studio Main window.

A separate panel appears at the bottom of the Main window. This is the Script Editor.

9. Enter the following function in the Script Editor:

```
function square(x)
{
    return x*x;
}
```

10. Click Apply in the Chart inspector.




The curve is represented as a straight line that is positioned at $y = 0$. This indicates that the function you have just written in the Script Editor has not been evaluated yet. To force evaluation just switch to Test Mode

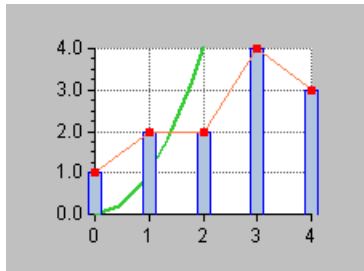
Switching to Test Mode

Note: To have access to Test Mode, you must have loaded the GUI application plug-in.


To force an evaluation by switching to Test Mode, do the following:

1. Click the Test icon  in the toolbar of the IBM ILOG Views Studio Main window.
A test panel opens containing your chart. In that panel, the curve of the square function is drawn correctly, which shows that the function written in the Script Editor has been evaluated.
2. Close the Test panel and return to your current working buffer window.
3. To force the redrawing of the chart, just click the chart.

You can now see the square function displayed as a curve in your regular chart.



Defining the Square-Root Function Data Set

You are going to repeat the same procedure to define the second data set and its script function. You can either use a data set that is already defined for the current chart or create another data set by simply clicking the Add icon  below the Data Sets list once the first data set is selected.

1. Select the second data set in the Data sets list.
2. Make sure Function is selected as the data set type.

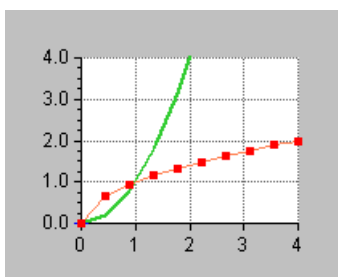
The values you set for Data count, X minimum, and X maximum are the same as those you specified for the square function data set. The Data count should be 10. The X minimum should be 0. The X maximum should be 4.

3. Type `squareRoot` in the Script function field.
4. Type `Square Root Function` in the Name field to change the data set name.
5. Add the following function in the Script Editor below the previous one:

```
function squareRoot(x)
{
    return Math.sqrt(x);
}
```

6. Click Apply in the Chart inspector.
7. Follow the steps described in the *Switching to Test Mode* on page 63 section to force an evaluation of the `squareRoot` script function.

The chart should look like this:





8. You can now choose to close the Script Editor by clicking the  icon.

Defining the Displayers

After you have defined the data sets to be represented, you need to define the displayers.

1. Click the Displayers tab in the Chart inspector to bring this page to the front.

The top of this page shows the list of displayers and the list of data sets created for the current chart. At this stage, you can either use a displayer that is already defined for the current chart or create another displayer by simply clicking the Insert icon  below the Displayers list.

Just remember to delete the displayers that you will not use since you only need two displayers for our example. To delete a displayer, select the displayer and click the Remove icon  below the Displayers list.


2. Make sure the first displayer is selected in the Displayers list.

This displayer will be used to display the `Square Function` data set.

3. For the displayer to display the `Square Function` data set, the toggle in front of the `Square Function` data set must be checked. If it is not checked, click the toggle.

Next, you must specify the type of displayer for the data set. Because you want to represent the square function with a polyline, you must specify a polyline displayer.

4. Make sure “Polyline” is selected in the Displayer type list. If it is not selected, select “Polyline.”
5. Click Apply to validate the change.

Following the same procedure, you are now going to define the displayer for the `Square Root Function` data set. You can either use a displayer that is already defined for the current chart or create another displayer by simply clicking the Add icon  below the Displayers list once the first displayer is selected.

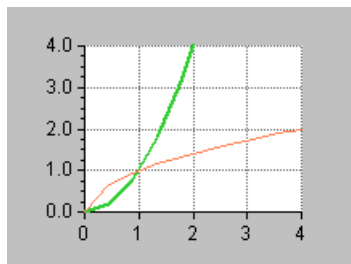
1. Select the second displayer from the Displayers list.

- For the displayer to display the Square Root Function data set, the toggle in front of the Square Root Function data set must be checked. If it is not checked, click the toggle.

Next, you must specify the type of displayer for the data set. Because you want to represent the square root function with a polyline, you must specify a polyline displayer.

- Make sure “Polyline” is selected in the Displayer type list. If it is not selected, select “Polyline.”
- Click Apply to validate the change.

You can now check the resulting chart. It should look like this.



Using Logarithmic Scales

This section explains how to use logarithmic scales. The procedure is based on the previous example where you created a Cartesian chart to represent the square and square root functions. Starting from that chart, you are going to customize the abscissa scale and the ordinate scale to obtain the following chart:

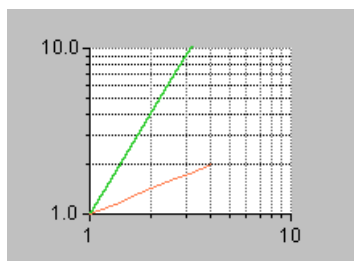


Figure 2.5 Using Logarithmic Scales

In this part of the example, you will see how to do the following tasks:

- ◆ Define a logarithmic transformation to be applied to the data values represented by a scale.
- ◆ Customize a grid associated with a scale.

Customizing the Abscissa Scale

To customize the abscissa scale for this example, you are going to apply a transformation to the data values represented by the scale. You are also going to customize the associated grid.

Applying a Transformation

1. Click the Scales tab in the Chart inspector to bring this page to the front.

Abscissa should be selected by default in the Scales list. The right part of the window is divided into several subpages that describe the properties of the selected scale.

2. Click the Transformation tab of the Scales page.

By default, no transformation is defined. Defining a transformation consists of applying an elementary transformation first, optionally followed by a logarithmic transformation. For this example, we want to apply only a logarithmic transformation so we are going to set the identity as the elementary transformation.

3. Select Identity in the Elementary transformation drop-down list.
4. Click “Logarithmic transformation” and enter 10 as the logarithmic base.
5. Click Apply to validate the changes.

Customizing the Grid

To customize the grid associated with the selected scale, do the following:

1. On the Scales page, click the Grid tab to bring this subpage to the front.

The Grid subpage is where you can customize the grid associated with the current scale.

2. Make sure the “Use a grid” toggle is checked.
3. To specify grid lines for both the major and the minor ticks, make sure the option “Draw major lines only” is deselected.
4. Click Apply to validate the changes.

Customizing the Ordinate Scale

To customize the ordinate scale for this example, you are going to apply a transformation to the data values represented by the scale. You are also going to customize the associated grid.

The Scales page of the Chart inspector should already be selected so you can continue with the following steps.

Applying a Transformation

1. Select Ordinate 1 in the Scales list of the Scales page.
2. Select the Transformation subpage.

This is where you are going to define the transformation you want to apply to the data.

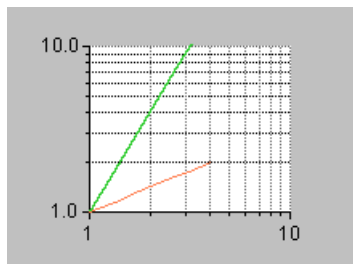
3. Select Identity in the Elementary transformation drop-down list.
4. Click “Logarithmic transformation” and enter 10 as the logarithmic base.
5. Click Apply to validate the changes.

Customizing the Grid

To customize the grid associated with the selected scale, do the following:

1. Select the Grid page on the Scales page.
This page lets you customize the grid associated with the current scale.
2. Make sure the “Use a grid” toggle is checked.
3. To specify grid lines for both the major and the minor ticks, make sure the option “Draw major lines only” is deselected.
4. Click the colored icon at the right end of the Minor palette field.
The Resources panel opens.
5. Choose “alternate” from the Line Style drop-down list.
6. Click Apply to validate the changes in the Resources panel and then Close to close the Resources panel.
7. Click Apply in the Chart inspector to validate the changes to the chart.

You can now check your resulting chart. It should look like this:

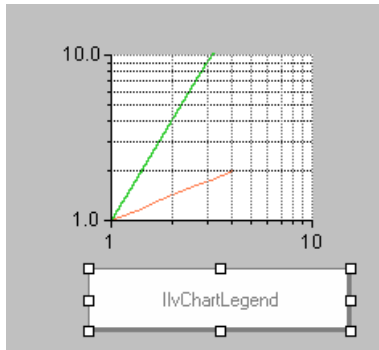


Connecting a Legend to a Chart

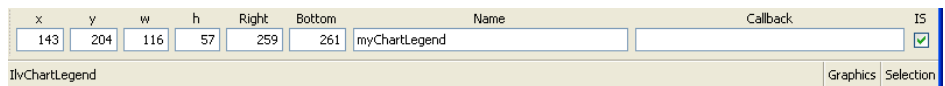
In this part of the example, you will see how to connect a legend to the chart that you have just created to represent the square and the square root functions (see *Using a Data Set Defined by a Script Function* on page 60). You would use this same procedure for any other legend and chart you might create.

To connect a legend to a given chart, do the following:

1. Drag the legend (instance of `IlvChartLegend`) from the Charts palette to your working buffer window.

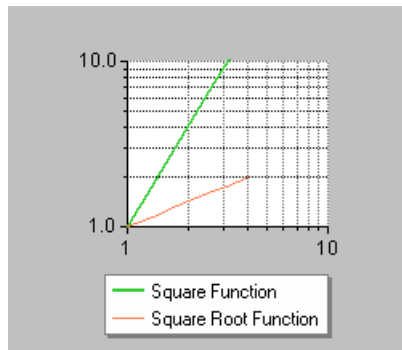


2. In the Name field of the Generic Inspector displayed at the bottom of the Main window, type `myChartLegend` to associate a name with the legend.



3. Double-click the chart to open the Chart inspector.
4. Click the Miscellaneous tab to bring this page to the front.
5. Select `myChartLegend` from the Legend drop-down list.
6. Click Apply to validate the change.

The legend is now connected to the chart. It shows two curve segments followed by the name of the data sets they represent. The name of the data sets appear because no specific text has been defined to illustrate the curves in the legend.



If you want to specify a legend text that is different from the name of the represented data set for a given curve, continue with the following steps:

7. Select the Displayers page of the Chart inspector.
8. Select each displayer and enter the text you want in the Legend field of the General subpage.
9. Click Apply to validate the changes.

This completes the example for charting analytic functions. In working through this example, you have learned how to do the following:

- ◆ Use data sets defined by Script functions in your charts.
- ◆ Use logarithmic scales in your charts.
- ◆ Connect a legend to a chart.

Using Polar Charts

In this chapter, you will learn how to use polar charts to represent in a circular way both angle values and non-angle values.

By default, a polar chart represents data expressed in polar coordinates (θ, ρ) in a circular way. The abscissa values θ are expressed in degrees and are mapped along a circular scale graduated in degrees by default. Therefore, you will use a polar chart to represent data whose abscissa values are expressed in degrees. However, a polar chart can also be used to get a circular representation of any data you want, even if the abscissa values are not expressed in degrees. To do this, you can do one of the following:

- ◆ Apply a transformation to the abscissa scale to convert the abscissa values into degrees. In this case, the abscissa scale will be graduated in degrees, no matter what the initial data may be.
- ◆ Indicate to the projector that the values to be projected are not expressed in degrees and then set a starting angle and a range to tell the projector how the abscissa values are mapped along the abscissa scale. In this case, the abscissa scale will bear the original abscissa values instead of being graduated in degrees.

In this chapter, you will see two examples that show how to represent data that are not expressed in degrees on a polar chart.

- ◆ *Example 1: Representing Values Expressed in Radians* shows how to represent values expressed in radians.
- ◆ *Example 2: Representing Time Values* shows how to represent time values.

Example 1: Representing Values Expressed in Radians

In this example, you will represent data in the form (θ, ρ) , where θ is expressed in radians, on a polar chart.

- ◆ In *Case 1*, you will apply a transformation to the abscissa scale to convert radians to degrees.
- ◆ In *Case 2*, you will set a starting angle and a range to map the radians along the abscissa scale.

Case 1: Applying a Transformation

In this example, you will perform the following tasks:

- ◆ *Creating a Polar Chart*
- ◆ *Defining Data with Abscissa Values in Radians*
- ◆ *Defining the Displayers*
- ◆ *Specifying the Transformation on the Abscissa*

Figure 3.1 shows the chart you are going to create.

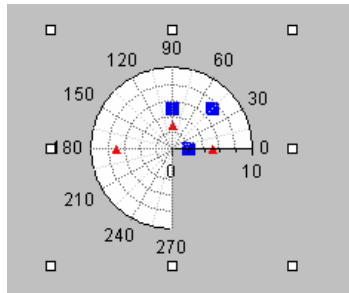


Figure 3.1 Polar Chart with Data in Radians on a Scale in Degrees

Creating a Polar Chart

Because our example chart is based on a polar chart, the first thing you must do is create the chart.

1. If you have not already done so, launch IBM ILOG Views Studio and display the Charts palette as described in *Launching IBM ILOG Views Studio with the Charts Extension* on page 16.
2. Drag a polar chart from the Charts palette to your working buffer.
3. Double-click this object to open its inspector.

Example 1: Representing Values Expressed in Radians

See *Using the Chart Inspector* on page 20 for an overview of the Chart inspector. This section contains an explanation of each notebook page you may need to use when working with charts.

4. At this stage, it is a good idea to save the buffer containing your chart.


Make sure you select a directory to which you have write access. Also remember to save periodically as you work.

Defining Data with Abscissa Values in Radians

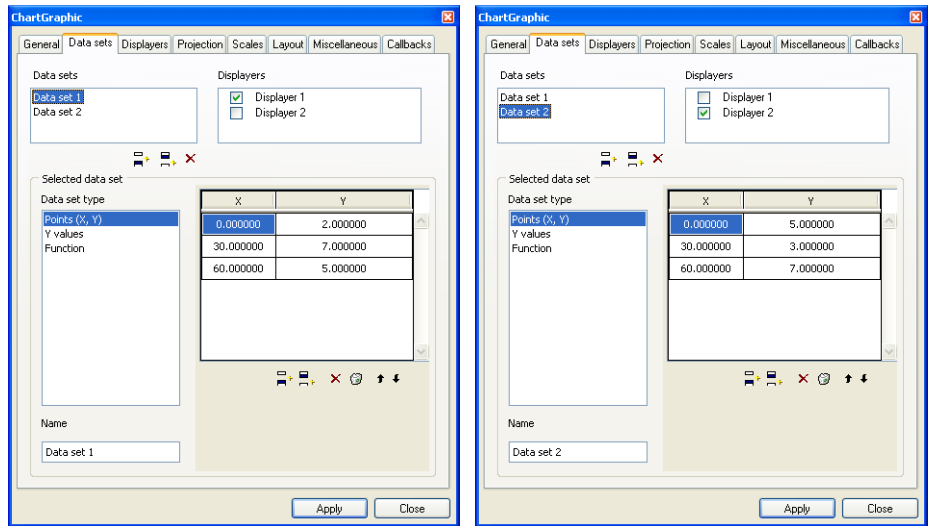
You are going to define two data sets, each consisting of three data points with two coordinates. The x-coordinate will be expressed in radians.

1. Click the Data sets tab in the Chart inspector to bring this page to the front.

To simplify the exercise, we are going to keep the data sets that are already defined. These data sets are of the type “Points (X,Y)” for the current chart. You can see that the abscissa values are expressed in degrees by default. For this exercise, you must therefore set the abscissa of the data points to radians in order to have data with abscissa values expressed in radians.

2. Select the first data set in the Data sets list.
3. Keep only the first three data points and delete the others. To delete a data point, just select it and click the Remove icon  below the table.
4. Click Apply to validate the changes.
5. Select the second data set and repeat step 3 and 4.

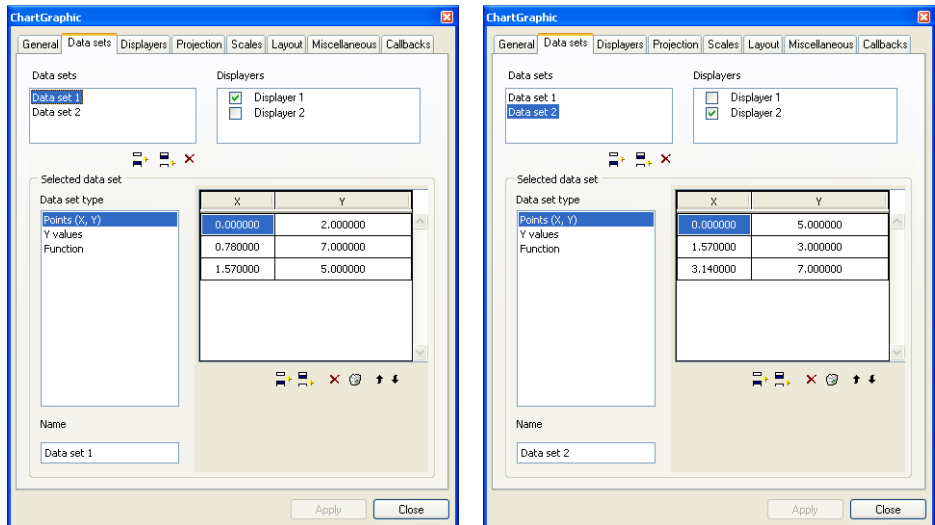
You have now two data sets with three data points each.



You are now going to select the abscissa value of each data point and replace the value in degrees by a value in radians.

6. Select the first data set again. Replace the first X value with 0, the second X value with 0.78, and the third X value with 1.57.
7. Click Apply to validate the changes.
8. Select the second data set. Replace the first X value with 0, the second X value with 1.57, and the third X value with 3.14.
9. Click Apply to validate the changes.

Example 1: Representing Values Expressed in Radians



Defining the Displayers

We want the two data sets to be represented by markers. To do so, the displayers displaying the data sets must be of the scatter type.

1. Click the Displayers tab to bring this page to the front.
2. Select the first displayer from the Displayers list.
3. Make sure “Scatter” is selected in the Displayer type list. If not, select “Scatter”.
4. Click Apply to validate the changes.
5. Repeat steps 2 to 4 for the second displayer.

Specifying the Transformation on the Abscissa

Next you are going to apply a transformation to the abscissa scale.

The abscissa values of your data points are now expressed in radians. To represent these data points on an abscissa scale graduated in degrees, you have to do the following:

- ◆ Specify the minimum and maximum values that will be represented by the scale.
- ◆ Apply a transformation to the scale to convert radians to degrees.

Do the following:

1. Select the Scales page of the Chart inspector.
2. Make sure Abscissa is selected in the Scales list.

3. Make sure the General page is displayed in front.

On this page, you are going to change the minimum and maximum data values represented on the scale because these values must be expressed in the same coordinate system as the data.

4. Enter 0 in the Min field and 4.71 in the Max field.
5. Click Apply to validate the changes.
6. Click the Transformation tab on the Scales page.

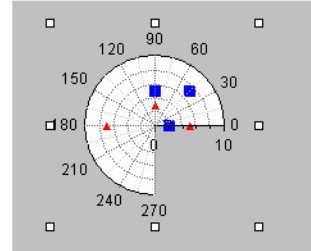
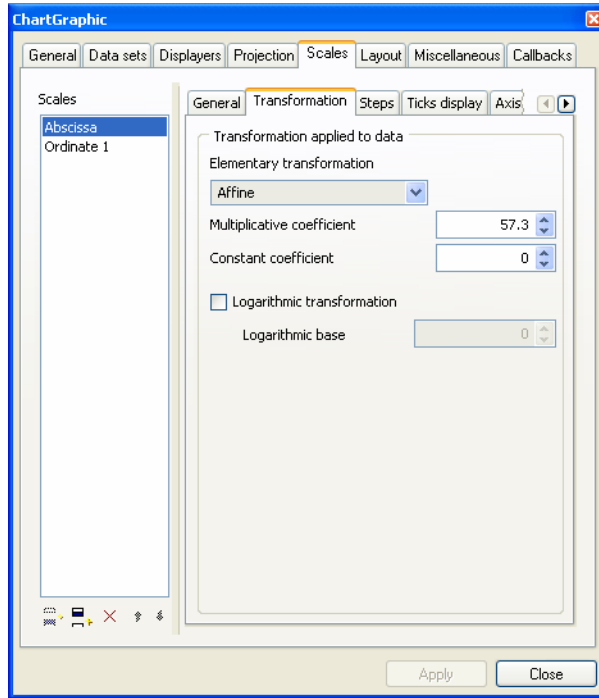
The initial abscissa values of the data are expressed in radians while the abscissa scale of a polar chart is graduated in degrees by default. You will use the transformation page to set the transformation that will convert the abscissa values from radians to degrees. This transformation is given by the following formula:

$$d = r \times \frac{180}{\pi}$$

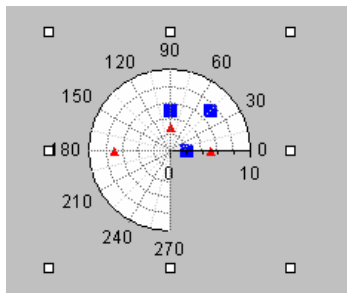
where r is the value expressed in radians and d the corresponding value expressed in degrees.

7. Select Affine in the “Elementary transformation” drop-down list.
8. Enter 57.3 as the multiplicative coefficient and 0 as the constant coefficient.
9. Click Apply to validate the changes.

Example 1: Representing Values Expressed in Radians



Your polar chart now represents your data expressed in radians on an abscissa scale graduated in degrees.



However, you may want to have an abscissa scale that is graduated in the same coordinate system as your data — in radians in this example. This case is detailed in the next section.

Case 2: Setting a Starting Angle and a Range

Case 2 assumes that you have already completed all the steps described in *Case 1*. From now on, you are going to work on a copy of the chart you obtained at the end of Case 1 so that you can compare the two charts at the end of the exercise. The two charts will appear as shown in Figure 3.2.

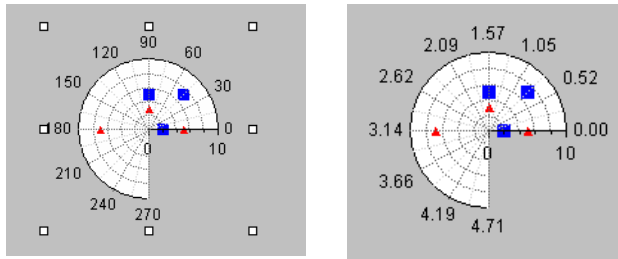


Figure 3.2 *Abscissa Scale in Degrees vs. Abscissa Scale in Radians*

Before beginning the tasks in this section, copy the chart you created in Case 1 and paste it either to the same buffer or to another buffer. (Click in the buffer before you choose Paste.)

To continue with this part of the example, you first need to remove the transformation previously set on the abscissa scale.

Disabling the Transformation

1. Double-click the polar chart to open its inspector.
2. Select the Scales page.
3. Make sure Abscissa is selected in the Scales list.
4. Select the Transformation page.
5. Choose None from the “Elementary transformation” drop-down list to disable the transformation.
6. Click Apply to validate the change.

All you have to do now is specify a starting angle and a range to tell the projector how to map the radians values along the abscissa scale.

Mapping Radians Values Along the Abscissa Scale

1. Select the Projection page of the Chart inspector.
2. Make sure the option “Projected values expressed in degrees” is deselected.
3. Make sure the Starting angle is set to 0.

Example 1: Representing Values Expressed in Radians

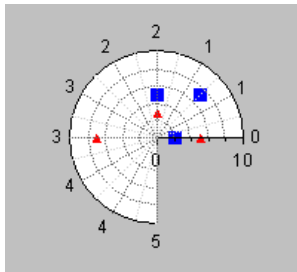
This means that you want the minimum value considered for the data values of the abscissa to be drawn at 0 degree.

4. Enter 270 in the Range field.

This means that you want the maximum value considered for the data values of the abscissa to be drawn at a 270-degree angle (= starting angle + range). You previously set 0 and 4.71 radians as the minimum and maximum data values considered for the abscissa scale. These values are equivalent to 0 and 270 degrees, respectively. (See step 4 of *Specifying the Transformation on the Abscissa* on page 76.)

5. Click Apply to validate the changes.

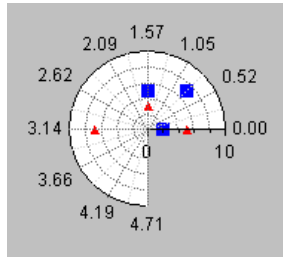
You should now have the same chart as in Case 1, except that the abscissa scale is graduated in radians.



Notice, however, that the scale labels are formatted as integers. To format them as floating values, do the following:

1. Select the Scales page of the Chart inspector.
2. Make sure Abscissa is selected in the Scales list.
3. Click the Steps tab of the Scales page.
4. Change the label format to "% .2f".
5. Click Apply to validate the change.

The resulting chart should now look like this. You can compare it with the chart you created in Case 1.



Summary

You have now completed the first example that shows how to use a polar chart to represent data whose abscissa values are not expressed in degrees. Although you changed the abscissa scale unit from degrees to radians, this example still displays data whose abscissa expresses angle values.

You can also use a polar chart to represent any data you want on a circular scale, even if the abscissa values are not directly related to angles. For example, a pie chart is based on a polar chart. For this reason, we are now going to show you another simple example that shows how to use a polar chart to represent time values in a circular way.

Example 2: Representing Time Values

You are now going to represent temperatures that have been recorded every two hours in the morning. The temperatures are in degrees Celsius and are represented by a marked polyline. Let's suppose you want to represent the time on a circular scale as on a clock. The following table lists the temperatures you are going to display in the chart:

Table 3.1 *Temperatures Recorded Every Two Hours*

Hour	Temperature
2	8
4	9
6	11
8	12
10	14
12	15

Figure 3.3 shows the chart that you will obtain.

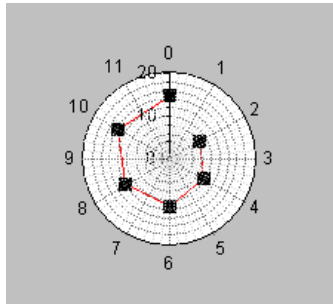


Figure 3.3 Polar Chart Representing Time Values

To build this chart, you will perform the following tasks:

- ◆ *Creating the Polar Chart*
- ◆ *Defining the Data Set*
- ◆ *Defining the Displayer*
- ◆ *Customizing the Projection*
- ◆ *Customizing the Abscissa Scale*
- ◆ *Customizing the Ordinate Scale*

Creating the Polar Chart

1. If you have not already done so, launch IBM® ILOG® Views Studio and display the Charts palette as described in *Launching IBM ILOG Views Studio with the Charts Extension* on page 16.
2. Drag a polar chart from the Charts palette to your working buffer.
3. Double-click this object to open its inspector.

See *Using the Chart Inspector* on page 20 for an overview of the Chart inspector. This section contains an explanation of each notebook page you may need to use when working with charts.


4. At this stage, it is a good idea to save the buffer containing your chart.


Make sure you select a directory to which you have write access. Also remember to save periodically as you work.

Defining the Data Set

You are now going to define a data set and enter the time and temperature values listed in Table 3.1.


1. Select the Data sets page of the Chart inspector.

The top of this page shows the list of data sets and the list of displayers created for the current chart. At this stage, you can either use a data set that is already defined for the current chart or create another data set by simply clicking the Insert icon  below the Data sets list.


Just remember to delete the data sets that you will not use since we only need one data set for our example. To delete a data set, select the data set and click the Remove icon  below the Data sets list.

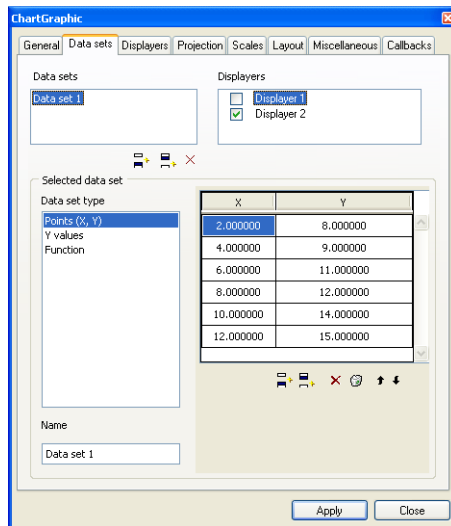
2. Make sure the first data set is selected in the Data sets list.

You are going to use this data set to represent the recorded temperatures. The time values will be displayed along the abscissa scale and the temperature values along the ordinate scale.

3. Make sure “Points (X,Y)” is selected in the Data set type list. If not, select “Points (X,Y)” in the Data set type list.
4. Make sure the (X,Y) list in the scrolling list next to the Data set type list is empty. If not, click the Clean icon  below the list to empty it.

You are now going to fill in the data set with the hours and the temperatures from Table 3.1.

5. Click the Add icon  below the empty (X,Y) list as many times as necessary to create the required number of empty rows.
6. Select the first cell (X cell) of the first row and enter the first time value.
7. Select the second cell (Y cell) of the first row and enter the first temperature value.
8. Repeat steps 6 and 7 for each row.
9. Click Apply to validate the changes.





10. Change the default name of the data set to `Morning Temperatures` in the Name field.
The Data sets list is updated accordingly.
11. Click Apply to validate the change.

Defining the Displayer

You are now going to set the type of the displayer used to display the data set representing the morning temperatures.

1. Select the Displayers page of the Chart inspector.

The top of this page displays the list of data sets and the list of displayers created for the current chart. At this stage, you can either use a displayer that is already defined for the current chart or create another displayer by simply clicking the Insert icon  below the Displayers list.

Just remember to delete the displayers that you will not use since we only need one displayer for our example. To delete a displayer, select the displayer and click the Remove icon  below the Displayers list.

2. Make sure the first displayer is selected in the Displayers list.
This displayer will be used to display the `Morning Temperatures` data set.
3. For the displayer to display the `Morning Temperatures` data set, the toggle in front of the `Morning Temperatures` data set must be checked. If it is not checked, click the toggle.

Next, you must specify the type of displayer for the data set. Because you want to represent the morning temperatures with a marked polyline, you must specify a marked polyline displayer.

4. Make sure “Marked polyline” is selected in the Displayer type list. If it is not selected, select “Marked polyline” in the Displayer type list.
5. Click Apply to validate the changes.

Now that you have defined the data set and the displayer you need, you are going to customize the projection.

Customizing the Projection

1. Select the Projection page of the Chart inspector.

This is where you specify a starting angle and a range that tells the projector how to map the hours along the abscissa scale.

2. Make sure the option “Projected values expressed in degrees” is deselected.
3. Enter 90 in the Starting angle field.

This means that you want the minimum value considered for the data abscissa values to be drawn at a 90-degree angle.

4. Enter 360 in the Range field.

This means that you want the maximum value considered for the data abscissa values to be drawn at a 450-degree angle (= starting angle + range). In other words, the abscissa values will be drawn along a full circle.

5. Make sure the option “Oriented clockwise” is selected.

The values will be displayed clockwise.

6. Click Apply to validate the changes.

To complete your polar chart, you need to customize both scales.

Customizing the Abscissa Scale

To customize the abscissa scale for this example, you need to perform the following tasks:

- ◆ *Specifying the Minimum and Maximum Data Values*
- ◆ *Setting the Graduations*

Specifying the Minimum and Maximum Data Values

1. Select the Scales page of the Chart inspector.

2. Make sure Abscissa is selected in the Scales list.

The right part of the Scales page displays several subpages that describe the properties of the selected scale.

3. Make sure the General subpage is displayed in front.

This is where you specify the minimum and maximum data values that will be represented by this scale.

4. Enter 0 in the Min field and 12 in the Max field.

These values are chosen because the minimum and maximum values must be expressed in the same coordinate system as the data and the abscissa scale should represent the hours as on a clock.

5. Click Apply to validate the changes.

Setting the Graduations

1. Click the Steps tab of the Scales page.
2. Make sure “Step number” is selected in the Step definition list.
3. Set 13 as the number of steps and 4 as the number of substeps between steps so that the abscissa scale resembles a clock.
4. Click Apply to validate the changes.

Customizing the Ordinate Scale

The last task of this exercise consists of customizing the ordinate scale. The Scales page should already be selected in the Chart inspector, so you can continue with this task.

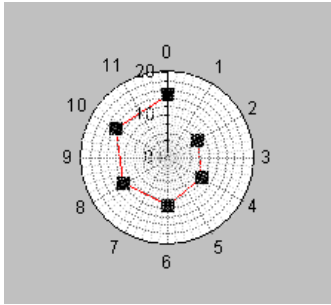
1. Select Ordinate 1 in the Scales list.
2. Click the General tab of the Scales page.

This is where you set the minimum and maximum data values that will be represented on this scale.

The minimum and maximum values must include all the data ordinate values, that is, all the recorded temperatures. These temperatures range from 8 to 15 degrees. The minimum value of the scale must therefore be no more than 8 and the maximum value no less than 15, if you want to make sure that all the temperature values in Table 3.1 are displayed. Moreover, it is common practice to use round values.

3. Enter 0 in the Min field and 20 in the Max field.
4. Click Apply to validate the changes.

The resulting chart should now appear as follows:



You have now completed all the examples to help you get started using IBM® ILOG® Views Studio with the Charts extension. You have learned the basic procedures for creating and customizing Cartesian and polar charts. You should be able to begin creating your own charts using IBM ILOG Views Studio.

In Part 2 of this manual, you will find detailed information on using the Charts Library.

Part II

Using the Charts Library

This part consists of the following chapters:

- ◆ Chapter 4, *Introducing the Charts Library* describes the main features of the Charts Library of IBM® ILOG® Views.
- ◆ Chapter 5, *Chart Basics* provides background information and describes the basic concepts upon which the Charts Library is based.
- ◆ Chapter 6, *Data Handling* describes the data handling features of the Charts Library.
- ◆ Chapter 7, *Chart Layout* describes how the layout of the charts is computed.
- ◆ Chapter 8, *Data Display* describes how data are displayed using the chart displayers of the Charts Library.
- ◆ Chapter 9, *Scales Display* describes how scales are displayed using the scale displayers of the Charts Library.
- ◆ Chapter 10, *Decorations Display* describes how to add legends, grids, and annotations to your charts.

- ◆ Chapter 11, *Interacting with Charts* describes how to use the interactors of the Charts Library.
- ◆ Chapter 12, *Using Charts to Display Real-Time Data* tells you how to use charts to display real-time data.

Introducing the Charts Library

The Charts Library of IBM® ILOG® Views is composed of dedicated classes that allow you to display your data in charts. Using the Charts Library of IBM ILOG Views, you will be able to produce many types of charts very easily, from the very simple to the very complex.

In this chapter, you will find information on the following topics:

- ◆ *Main Features of the Charts Library*
- ◆ *Feature Illustrations*

Main Features of the Charts Library

The Charts Library allows you to display data in charts that can be customized in various ways and to interact with these charts in different manners.

Global Chart Characteristics

The following are the global characteristics of the Charts Library of IBM® ILOG® Views:

- ◆ The Charts Library has been designed with a clear separation between the data and the graphical representations of the data.
- ◆ A chart can display as many graphical representations of data as you want.

- ◆ A chart uses one abscissa scale and as many ordinate scales as you want.
- ◆ Two types of charts are available:
 - *Cartesian charts* displaying data expressed in a *Cartesian system* of coordinates in a standard way.
 - *Polar charts* displaying data expressed in a *polar system* of coordinates in a circular way.
- ◆ The orientation of the scales can be customized.
 - In Cartesian charts, the abscissa and ordinate scales can be oriented either horizontally or vertically.
 - In polar charts, the circular abscissa scale can be oriented clockwise or counterclockwise and the radial ordinate scales can be oriented at any angle.
- ◆ The charts are *data-aware*. Changes made to data are automatically reflected in the charts that display these data. The possible modifications made by interacting with a chart are also automatically reflected on the data.
- ◆ Three *automatic scroll modes* (stop, shift, cyclic) are available. These modes allow you to define the behavior of a chart when new data that must be displayed are added to the chart.
- ◆ Charts can be created and customized without having to code using IBM ILOG Views Studio.

Data Features

The data to be represented in a chart can have different forms:

- ◆ Sets of data points with two coordinates
- ◆ Sets of data values
- ◆ Functions of the type $y = f(x)$ that can be described by a script or a C++ callback

The data can be expressed in:

- ◆ A Cartesian system of coordinates (x, y)
- ◆ A polar system of coordinates (θ, ρ)

The abscissa values (x or θ) are plotted along the abscissa scale and the ordinate values (y or ρ) are plotted along the ordinate scale(s).

Graphical Representations of Data

The following predefined graphical representations of data are available:

- ◆ Scatter (data displayed with markers)
- ◆ Polyline, polygon, and marked polyline (polyline with markers)
- ◆ Bar and 3D bar
- ◆ Step (data displayed with only the steps of the stairs) and stair (data displayed with the entire stairs)
- ◆ Fixed step (data displayed with fixed steps centered on the data)
- ◆ Stock representations including high-low representation made of lines, high-low representations made of bars, high-low open-close representations
- ◆ Bubble (data displayed with bubbles)
- ◆ Pie charts
- ◆ Side-by-side bars
- ◆ Stacked representations (100% stacked or not) including stacked bars, stacked 3D bars, stacked polygons

Scale Features

The following types of scales are available in the library:

- ◆ Rectangular scales (that is, scales with straight-line axes).
- ◆ Circular scales (can be used only in polar charts)

The scales can be graduated with:

- ◆ Linear graduations
- ◆ Logarithmic graduations

Decorations

The following kinds of decorations can be added to a chart to improve its appearance or to help in understanding the data:

- ◆ Legends that explain the displayed data
- ◆ Grids to help you locate the data points on a chart
- ◆ Cursors that show the values of a given data point on the scales of a chart
- ◆ Annotations linked to given data points

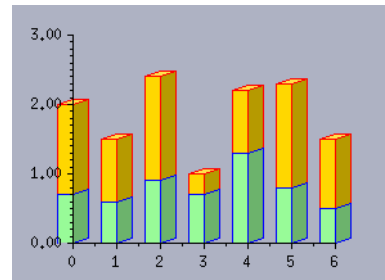
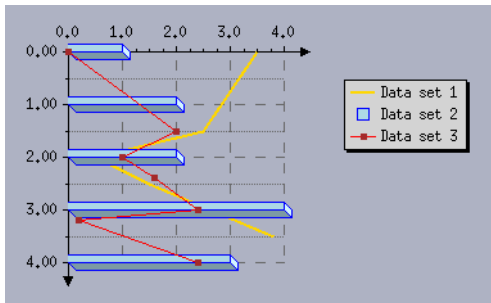
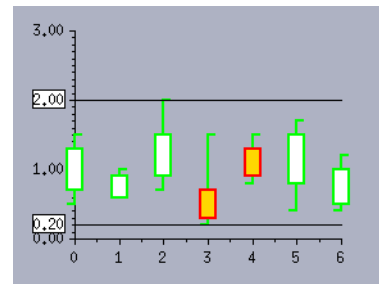
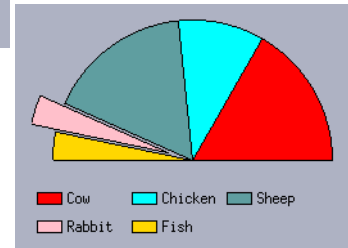
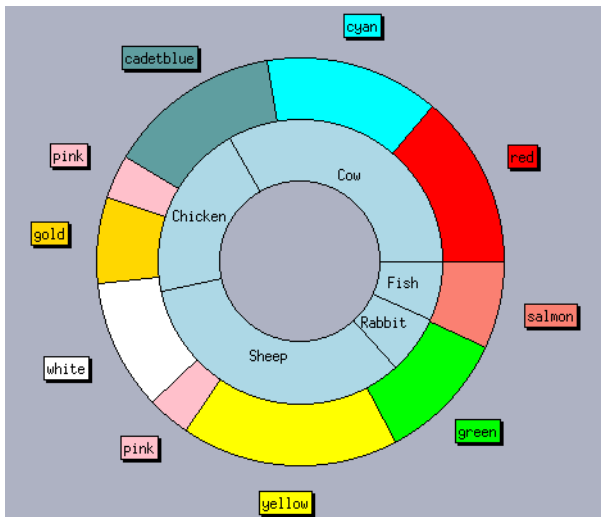
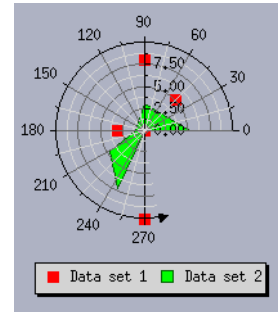
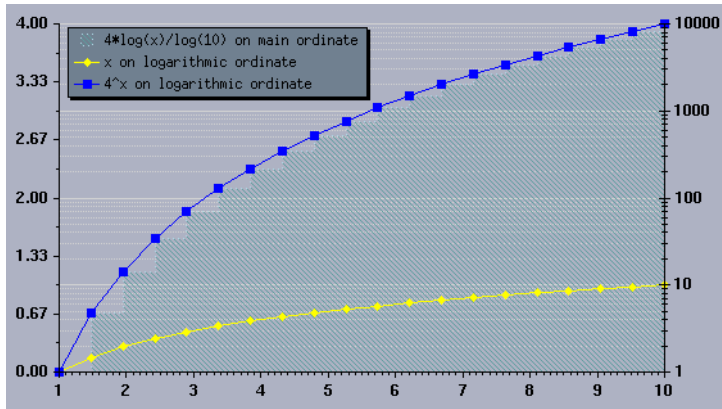
Grids, cursors, and annotations are all part of the chart. A legend is a separate graphic object that can be positioned independently of the chart.

Interactors

The following interactors are available that allow the user to interact with a chart.

- ◆ A zoom interactor to zoom and unzoom within the displayed data
- ◆ A scroll interactor to scroll the displayed data with the arrow keys
- ◆ A pan interactor to scroll the displayed data with the mouse
- ◆ A crosshair interactor that displays a crosshair at the location of the mouse pointer
- ◆ Data interactors that allow the user to do the following:
 - Drag a data point.
 - Highlight or display information related to a data point.
 - Select data points (either all the data points of a data set or only a single data point).

Feature Illustrations



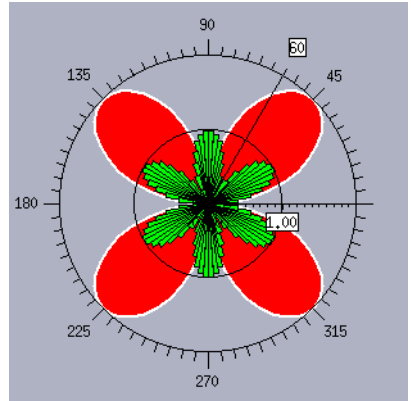
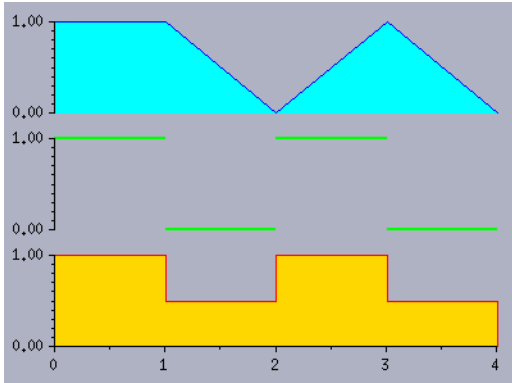


Chart Basics

This chapter presents the basic concepts that you will need to begin to display your data in charts and to understand how charts work. You will find information on the following topics:

- ◆ *What is a Chart?*
- ◆ *General Architecture of the Charts Library*
- ◆ *Basic Steps for Creating a Chart*
- ◆ *How Charts Work in IBM ILOG Views*

What is a Chart?

A *chart* represents data graphically in different forms (markers, lines, bars, and so on) with scales that are added to indicate the values of the displayed data. We distinguish two kinds of charts:

- ◆ **Cartesian charts** represent data in a standard way. The data is expressed using a Cartesian system of coordinates (x, y). The x - and y -coordinates are plotted along the abscissa and ordinate scales, respectively. The scales are *rectangular* and are displayed orthogonally.

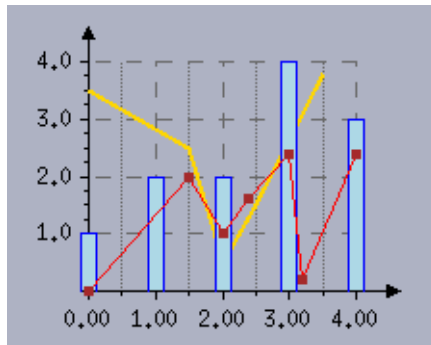


Figure 5.1 A Cartesian Chart

- ◆ **Polar charts** represent data in a circular way. The data is expressed using a polar system of coordinates (θ, ρ) . The abscissa values θ are plotted along a circular scale. The ordinate scale, along which the ρ -coordinates are plotted, is *rectangular* and is displayed radially.

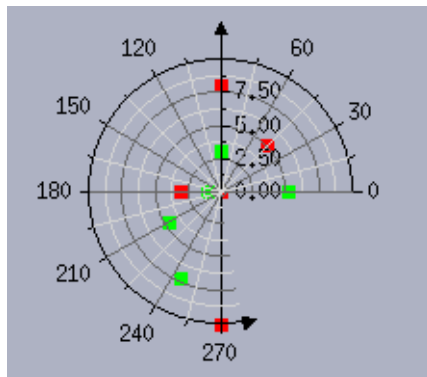


Figure 5.2 A Polar Chart

In addition, other elements can be added to a chart to aid in the understanding of the displayed data:

- ◆ Legends that explain the displayed data (see Figure 5.3)
- ◆ Grids to help you locate the data points on a chart (see Figure 5.3)
- ◆ Cursors that show the values of a given data point on the scales (see Figure 5.4)
- ◆ Annotation linked to a given data point (see Figure 5.5)

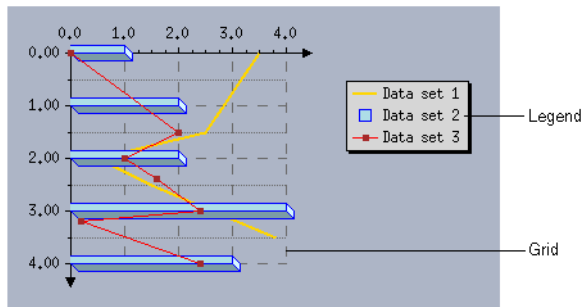


Figure 5.3 A Grid and a Legend in a Cartesian Chart

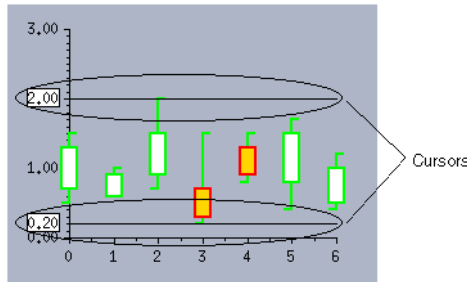


Figure 5.4 Cursors in a Cartesian Chart

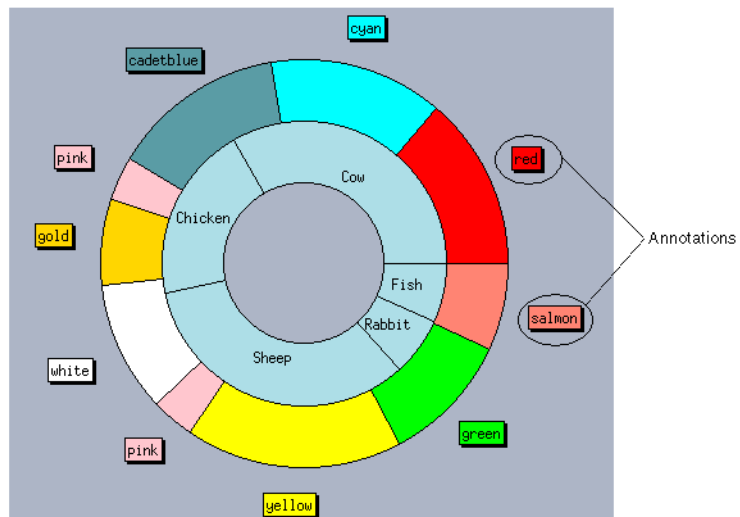


Figure 5.5 Annotations on a Pie Chart

General Architecture of the Charts Library

The general architecture of the Charts Library in IBM® ILOG® Views is based on the *model-view separation* concept. The Charts Library has been designed with a clear separation between the data and the graphical representations of the data. There is a clear distinction between:

- ◆ the *chart data objects* that manage the sets of data (or *data sets*) to be displayed by the charts and
- ◆ the *chart objects* that display chart data objects.

Listeners are set on data so that the chart objects can be updated automatically when modifications are made to the data they display.

Basic Rules

The following basic rules apply to chart data objects and chart objects:

- ◆ The same chart data object can be displayed by several chart objects.
- ◆ A given chart object can display only one chart data object.
- ◆ A given *data set* managed by a chart data object can be displayed with several different graphical representations in a chart object.

Example Charts

The two charts shown in Figure 5.6 will help you understand these basic rules. Each chart displays a set of mean temperatures for the morning and a set of mean temperatures for the afternoon recorded for each day of a week. However, the charts use a different type of graphical representation to display the same data. These two charts display the same chart data object that manages the two sets of temperatures.

- ◆ Chart 1 displays the morning and afternoon mean temperatures with two polylines. The two sets of temperatures are also displayed on the chart with high-low bars to illustrate the variation of the temperatures between the morning and the afternoon of each day of the week.
- ◆ Chart 2 displays the two sets of temperatures with stacked bars in order to illustrate the mean temperature for each day of the week.

In both charts, the abscissa scale indicates the days of the week and the ordinate scale the temperatures in degrees Celsius.

The drawing of the graphical representations of data (polylines, high-low bars, and so on) is performed within a chart by dedicated objects called *displayers*. Figure 5.6 shows the displayers used by the two charts to represent the morning and afternoon mean temperatures data sets. The solid-line arrows indicate the data set(s) that each displayer displays. The dotted-line arrows indicate the graphical representation that is displayed by each displayer.

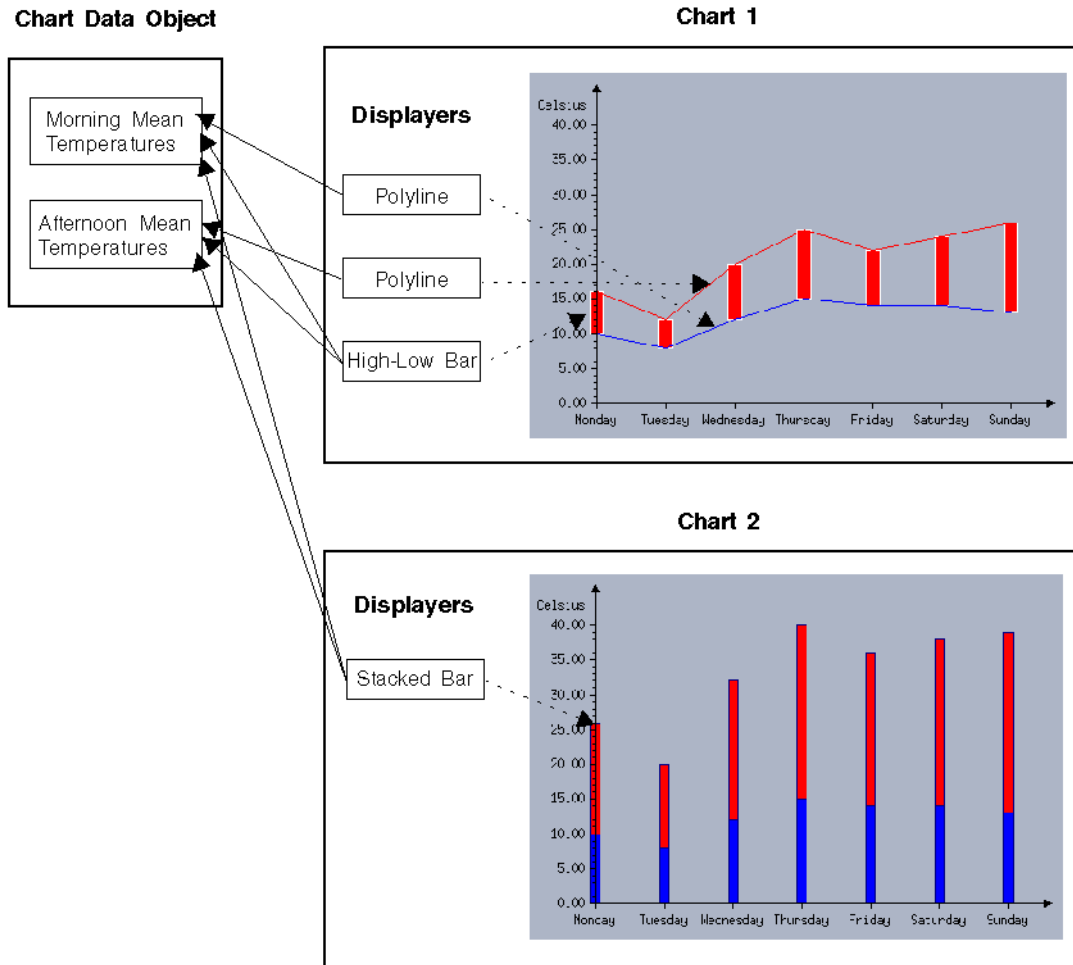


Figure 5.6 Two Charts using the Same Chart Data Object

Data Classes

The base class used to represent a chart data object is the `IlvAbstractChartData` class. A subclass called `IlvMemoryChartData` that stores the data sets it manages in memory is provided in the library.

The data sets managed by a chart data object are all instances of subclasses of the `IlvChartDataSet` class. Several subclasses of the `IlvChartDataSet` class are provided

to allow you to define a set of data to be displayed by a chart in different ways. A data set can be defined as:

- ◆ A set of points with two coordinates

This kind of data set is represented by an instance of the `IlvChartPointSet` class. The data points managed by this data set are instances of the `IlvDoublePoint` class that allows you to represent a point with two coordinates of the `IlvDouble` type.

- ◆ A set of values

This kind of data set is represented by an instance of the `IlvChartYValueSet` class. The data values managed by this data set are of the `IlvDouble` type. These values correspond to the ordinates of the data points that will be displayed. The abscissa of the data points are by definition the indexes of the stored values.

- ◆ A function

This kind of data set is represented by an instance of a subclass of the `IlvAbstractChartFunction` class. Two subclasses are available in the library: `IlvCallbackChartFunction` for which the function is defined by a C++ callback and `IlvScriptChartFunction` for which the function is defined by a script.

Chart Classes

The base class used to display a chart is the `IlvChartGraphic` class. This class defines a chart object as a graphic object that can be used in the same way as all the other graphic objects defined in IBM® ILOG® Views.

When you create a chart object, an `IlvMemoryChartData` object (inherited from `IlvAbstractChartData`) is created by default and is set on the created chart to handle the storage of the data sets it is going to display. You can either use this chart data object created by default or set your own chart data object.

The way the data are projected into screen coordinates is not specified at the level of the `IlvChartGraphic` class since it depends on the system of coordinates in which the data to be displayed are expressed. No object is defined by default to display the scales since the type depends on the way the data are projected into screen coordinates. Two subclasses of the `IlvChartGraphic` class are provided. These subclasses predefine the way the data are projected and the objects to be used to display the scales.

- ◆ The `IlvCartesianChart` class is instantiated to represent data in a standard way. The data are expressed in a Cartesian system of coordinates (x, y). Two rectangular scales are created by default: one representing the abscissa values x and one representing the ordinate values y .

- ◆ The `IlvPolarChart` class is instantiated to represent data in a circular way. The data are expressed in a polar system of coordinates (θ , ρ). A circular scale is created by default to represent the abscissa values θ . A rectangular scale is created by default to represent the ordinate values ρ .

By using one of these subclasses instead of the `IlvChartGraphic` class, you only need to set the following objects when you create a chart:

- ◆ The data sets to be displayed with the considered chart.
- ◆ The displayers used to draw the graphical representations of the data sets in the considered chart.

Basic Steps for Creating a Chart

The basic steps for creating a chart are the same whether you are creating a Cartesian chart or a polar chart. To create a chart, do the following:

1. Create the data sets.

- Create the objects representing the data sets you want to display. These objects are instances of subclasses of the `IlvChartDataSet` class (See *Data Classes* on page 100).
- Put the data to be displayed into the created data sets.

2. Create the chart.

The created chart object is an instance of the `IlvCartesianChart` or the `IlvPolarChart` class, depending on the type of chart you want to display (See *Chart Classes* on page 101).

3. Add the data sets to the chart data object.

The chart data object can be the `IlvMemoryChartData` object that is created by default or whatever chart data object you have set on the created chart.

4. Create and add the displayers that will be used to draw the graphical representations of the data sets.

This section provides two examples to show you how to create a Cartesian chart and a polar chart. These examples follow our basic steps for creating a chart.

- ◆ *Creating a Simple Cartesian Chart*
- ◆ *Creating a Simple Polar Chart*

Creating a Simple Cartesian Chart

You are going to see how to create a simple Cartesian chart that is similar to Chart 1 in Figure 5.6. The data to be represented are the morning and afternoon mean temperatures recorded for each day of a week. The temperatures are expressed in degrees Celsius. The days of the week are referenced by indexes from 0 to 6. The data for the chart are listed in Table 5.1. You will see the steps required to display these data in a Cartesian chart.

Table 5.1 Data for the Example Chart

Day	Morning Mean Temperature (° C)	Afternoon Mean Temperature (° C)
0	10	16
1	8	12
2	12	20
3	15	25
4	14	22
5	14	24
6	13	26

Figure 5.7 shows the Temperatures Chart that will be created to display our data. The morning mean temperatures will be displayed with a blue polyline (the bottom line of the chart) and the afternoon mean temperatures with a red polyline (the top line of the chart). In the chart, we also want to display the two sets of temperatures with high-low bars in order to illustrate the variation between the morning and afternoon temperatures for each day of the week. After the chart is constructed, we will show you how to add the legend that appears at the bottom of the chart.

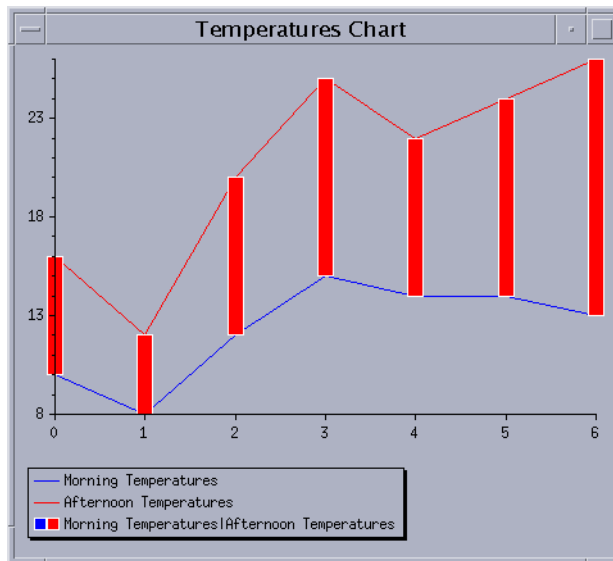


Figure 5.7 Example Cartesian Chart

The complete source code of this example can be found in the `cartesian.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory.

Creating the Data Sets

1. Create the objects representing the data sets we want to display.

We want to display two data sets on the same chart: the morning mean temperatures and the afternoon mean temperatures for each day of a week. The days will be plotted along the abscissa scale and the temperatures along the ordinate scale. To define each data set, we create a data set that stores data points with two coordinates (x, y).

```
//== Create two data sets.
IlvChartDataSet* dataSets[2];
//== Create one data set to store the morning temperatures.
dataSets[0] = new IlvChartPointSet("Morning Temperatures");

//== Create one data set to store the afternoon temperatures.
dataSets[1] = new IlvChartPointSet("Afternoon Temperatures");
```

2. Put the data to be displayed into the created data sets.

The day is set as the abscissa and the temperature as the ordinate of the stored data points.

```
//== Put the data into the morning temperatures data set.
dataSets[0]->addPoint(IlvDoublePoint(0, 10));
dataSets[0]->addPoint(IlvDoublePoint(1, 8));
dataSets[0]->addPoint(IlvDoublePoint(2, 12));
dataSets[0]->addPoint(IlvDoublePoint(3, 15));
dataSets[0]->addPoint(IlvDoublePoint(4, 14));
dataSets[0]->addPoint(IlvDoublePoint(5, 14));
dataSets[0]->addPoint(IlvDoublePoint(6, 13));

//== Put the data into the afternoon temperatures data set.
dataSets[1]->addPoint(IlvDoublePoint(0, 16));
dataSets[1]->addPoint(IlvDoublePoint(1, 12));
dataSets[1]->addPoint(IlvDoublePoint(2, 20));
dataSets[1]->addPoint(IlvDoublePoint(3, 25));
dataSets[1]->addPoint(IlvDoublePoint(4, 22));
dataSets[1]->addPoint(IlvDoublePoint(5, 24));
dataSets[1]->addPoint(IlvDoublePoint(6, 26));
```

Creating a Cartesian Chart

To create a Cartesian chart, use the following code:

```
IlvChartGraphic* chart = new IlvCartesianChart(display,
                                               IlvRect(10, 10, 450, 300));
```

The `IlvRect` object passed as a parameter specifies the bounding box of the created chart.

Adding the Data Sets to the Chart Data Object

The data sets are added to the chart data object set on the chart object. The chart data object is obtained for a given chart using the `IlvChartGraphic::getData` method.

```
IlUInt dataSetsCount = 2;
chart->getData()->setDataSets(dataSetsCount, dataSets);
```

Creating and Adding the Displayers

1. Create and add the objects used to display the graphical representations of the data.

We want to display the two sets of temperatures with two polylines and also with a high-low bar representation. The first polyline represents the morning mean temperatures data set and corresponds to `dataSets[0]`. The second polyline represents the afternoon

mean temperatures data set which corresponds to `dataSets[1]`. The high-low bar representation represents both the morning and afternoon mean temperatures data sets.

```
//== Create and add the displayers.
chart->addDisplayer(new IlvPolylineChartDisplayer(),dataSets[0]);
chart->addDisplayer(new IlvPolylineChartDisplayer(),dataSets[1]);

IlvPalette* risePalette = display->getPalette(display->getColor("red"),
                                             display->getColor("white"));
IlvPalette* fallPalette = display->getPalette(display->getColor("blue"),
                                             display->getColor("white"));

chart->addDisplayer(new IlvHiLoBarChartDisplayer(12,
                                             risePalette,
                                             fallPalette),
                  dataSetCount,
                  dataSets);
```

The first parameter of the constructor of the `IlvHiLoBarChartDisplayer` object, that is, 12 is the width of the bars. The `risePalette` is used to draw the high-low items for which the corresponding low value (that is, the morning mean temperature) is smaller than the high value (that is, the afternoon mean temperature). The `fallPalette` is used to draw the high-low items for which the corresponding low value is greater than the high value.

2. Set the colors for the polyline displayers.

We want the morning mean temperatures to be displayed in blue and the afternoon mean temperatures in red.

```
chart->getDisplayer(0)->setForeground(display->getColor("blue"));
chart->getDisplayer(1)->setForeground(display->getColor("red"));
```

Note: *The corresponding palettes could have been passed directly as a parameter to the constructors of the polyline displayers as has been done for the high-low bar displayer.*

We have now completed the basic steps for creating a chart. Additional steps can be performed to enhance the appearance of a chart. You are now going to see how to customize the abscissa and ordinate scales and how to add a legend to a chart.

Customizing the Abscissa Scale

We are now going to set the number of steps and substeps displayed on the abscissa scale. The steps are marked by *major tick marks* on the scale and the substeps are marked by *minor tick marks* that are drawn between the major tick marks.

The abscissa scale represents the days of the week. Since the number of days in a week is seven, the number of steps should be set to 7 so that one major tick mark appears for each day of the week. We do not want any minor tick marks between the major tick marks so we will set the number of substeps between two steps to 0.

The computation of the steps and substeps for a given scale is performed by a dedicated object called *scale steps updater* that is set on the scale. Since we want steps and substeps with a constant spacing, we first set a *constant* scale steps updater on the scale.

```
IlvSingleScaleDisplayer* abscissaScale = chart->getAbscissaScale();

IlvConstantScaleStepsUpdater* updater =
    new IlvConstantScaleStepsUpdater(abscissaScale);
delete IlvScaleStepsUpdater::Set(abscissaScale, updater);
```

Then we set on the scale steps updater the number of steps and substeps between two steps.

```
updater->fixStepsCount(7, 0);
```

Customizing the Ordinate Scale

We are now going to set the step and the substep units for the ordinate scale. Again, the steps will be marked with a major tick mark and the substeps with a minor tick mark.

The ordinate scale represents the temperature in degrees Celsius. We want a major tick mark to appear every five degrees, and a minor tick mark every degree. To do this, we specify 5 as the step unit and 1 as the substep unit.

Since we want steps and substeps with a constant spacing, we first set a *constant* scale steps updater on the scale.

```
IlvSingleScaleDisplayer* ordinateScale = chart->getOrdinateSingleScale();

updater = new IlvConstantScaleStepsUpdater(ordinateScale);
delete IlvScaleStepsUpdater::Set(ordinateScale, updater);
```

Then we set on the scale steps updater the step and the substep units.

```
updater->fixStepUnit(5, 1);
```

The Temperatures Chart now appears as shown in Figure 5.8.

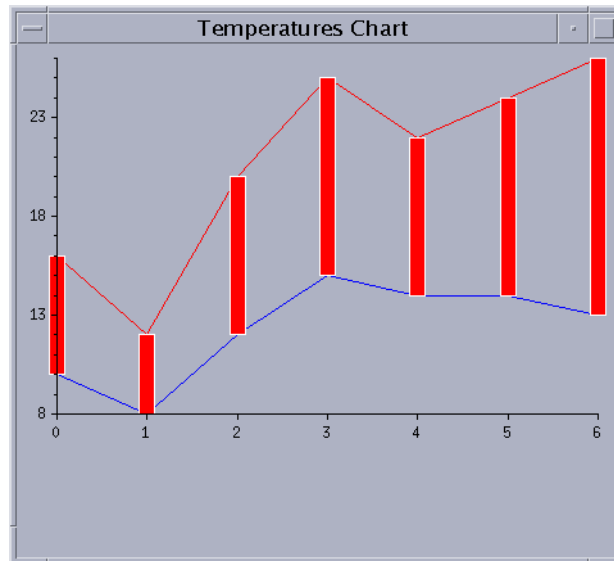


Figure 5.8 The Temperatures Chart

Adding a Legend

To add a legend to a chart, perform the following steps:

1. Create a chart legend object.

```
IlvChartLegend* legend = new IlvChartLegend(display,
                                           IlvRect(10, 330, 450, 50));
```

The chart legend object is a graphic object that is positioned independently of the chart. The `IlvRect` object passed as a parameter to the constructor gives the position and the default size of the legend. The created legend does not contain any legend items since it is not yet connected to a chart. A legend must be connected to a chart to have the legend items appear.

2. Connect the legend to the chart.

```
chart->setLegend(legend);
```

When the legend is set on the chart, the legend items corresponding to the displayers that are defined in the chart are automatically computed and the size of the legend is recomputed to fit these legend items.

Note: The chart legend object is a graphic object like the chart object and is positioned independently of the chart object. Just as you add the chart object to a container or manager, you must also add the chart legend object to the container or manager. Otherwise, the chart legend object will not appear on the screen.

Now, with the legend added, the Temperatures Chart appears as follows:

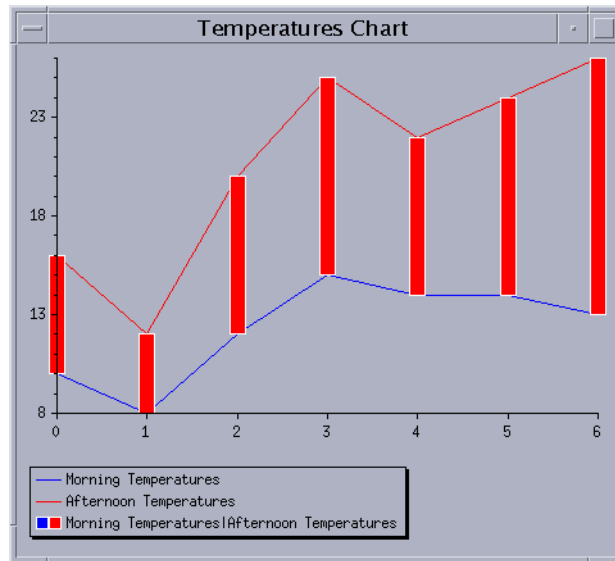


Figure 5.9 The Temperatures Chart with a Legend

Creating a Simple Polar Chart

You are now going to see how to create a polar chart. We will use the same data that we used for creating the Temperatures Chart in the previous section. Instead of representing the morning and afternoon mean temperatures in a Cartesian chart, we are going to represent them in a polar chart. We will follow the same basic steps to create a polar chart as we did to create our Cartesian chart.

For our polar chart, the data sets are the same, since we want to represent the same data as in the Cartesian chart. The days of the week will still be represented along the abscissa and the temperatures along the ordinate. The data will simply be displayed differently since the abscissa values will be mapped along a circular scale and the ordinate values will be displayed radially. The displays are also the same since we want to represent the data with the same graphical representations. We will customize the scales in the same way as we did for the Cartesian chart. However, we will show you some additional ways to customize the abscissa scale. Finally, we will also add a legend to the chart.

Figure 5.10 shows the polar chart that we will create.

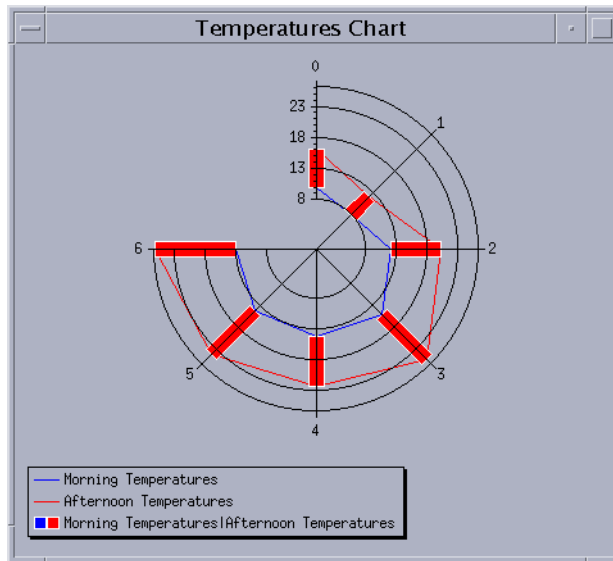


Figure 5.10 Example Polar Chart

The complete source code of this example can be found in the `polar.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory.

Creating the Data Set

Once again, the first step is to create the data sets and put the data to be displayed into the data sets. Use the same data and procedures as for the Cartesian chart example. See *Creating the Data Sets* on page 104.

Creating a Polar Chart

To create our polar chart, we use the following code:

```
IlvChartGraphic* chart = new IlvPolarChart(display,
                                           IlvRect(10, 10, 450, 300),
                                           IlvTrue,
                                           90,
                                           270,
                                           IlvTrue);
```

The `IlvRect` object passed as a parameter corresponds to the bounding box of the created chart. The Boolean value `IlvTrue` that follows the `IlvRect` parameter indicates that grids should be added to the scales created by default. The next parameter is set to 90 degrees and indicates the starting angle at which the minimum data value will be displayed. This means that the first day of the week referenced by the index 0 will be displayed at 90 degrees on the abscissa scale. The next parameter is set to 270 degrees and indicates the angle range within which the data will be projected on the screen. The last Boolean value indicates whether the

polar system of coordinates is oriented clockwise. It is set to `IlvTrue`, which means that the abscissa values will be displayed clockwise.

Note: The `range` parameter of the constructor of a polar chart must be set to the angle range within which you want the data to be projected on the screen if the abscissa values of the data are not expressed in degrees. Otherwise, the `range` parameter must be set to 0.

Adding the Data Sets to the Chart Data Object

To add the data sets to the chart data object, use the same data and procedures as for the Cartesian chart example. See *Adding the Data Sets to the Chart Data Object* on page 105.

Creating and Adding the Displayers

Because we want to use the same graphical representations for our data, we will use the same displayers as we used for the Cartesian chart. See *Creating and Adding the Displayers* on page 105.

We have now completed the basic steps for creating a chart. We will continue with the additional steps to customize the ordinate and abscissa scales and to add a legend to the chart.

Customizing the Ordinate Scale

For the ordinate scale, we want to customize the scale as we did for the Cartesian chart. See *Customizing the Ordinate Scale* on page 107.

Customizing the Abscissa Scale

We want to customize the abscissa scale as we did for the Cartesian chart. In addition, we want to change the format of the step labels along the abscissa scale and specify that the step labels should be drawn at the axes crossings for the abscissa scale.

1. First, we want to customize the scale so that a major tick mark appears for each day of the week. See *Customizing the Abscissa Scale* on page 106.
2. Change the format of the step labels.

The format used to display the labels at the major tick marks of the abscissa scale of a polar chart is set by default to `"%.2f"`. This displays the labels as floating values with two digits after the decimal point. Since the abscissa values are integers corresponding to the indexes of the days of the week, we want to set the step label format to `"%.0f"`. This will display the indexes of the days in the form of integers.

```
abscissaScale->setStepLabelFormat("%.0f");
```

3. Display the step labels at the intersection of the axes.

By default, the flag indicating whether the step labels must be drawn where the axes intersect is set to `IlvFalse` for the abscissa scale of a polar chart. The label "0" for the

first day of the week will not be drawn by default since the corresponding tick mark is located on the abscissa scale at the intersection of the ordinate scale. To make this label appear, the flag must be set to `IlvTrue`.

```
abscissaScale->drawLabelOnCrossings (IlvTrue);
```

Adding the Legend

Again, we want to add a legend to our chart. Use the same procedures as for the Cartesian chart example. See *Adding a Legend* on page 108.

Figure 5.11 shows our finished polar chart.

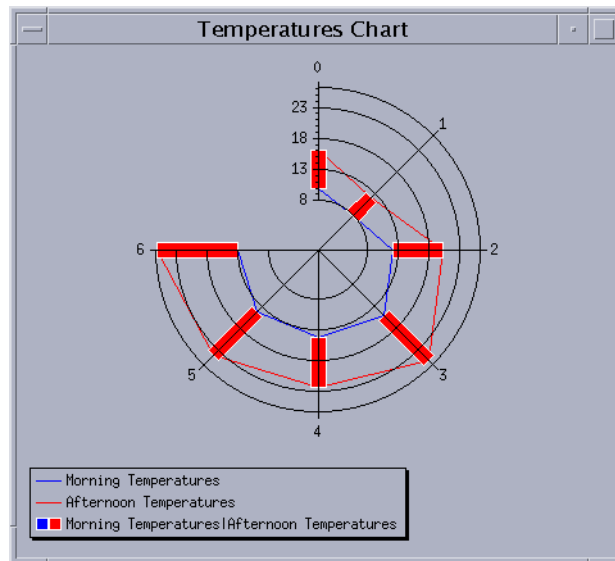


Figure 5.11 The Temperatures Chart Displayed as a Polar Chart

Additional Ways to Customize a Chart

The example charts in Figure 5.9 and Figure 5.11 are quite basic. At this point, we can customize the charts in several other ways to improve their appearance. The same improvements can be made to both of our charts. For example, we can do the following:

- ◆ Replace the values (0, 1, 2, and so on) displayed along the abscissa scale with text labels.
- ◆ Add a label at the end of the ordinate scale.
- ◆ Add an arrow at the end of each axis.

The complete source code of the examples can be found in the `cartesian_custom.cpp` file for the Cartesian chart and in the `polar_custom.cpp` file for the polar chart. These files are located in the `$ILVHOME/samples/charts/userman/src` directory.

Replacing the Values Displayed along the Abscissa Scale

By default, the labels displayed at the major tick marks of the scales are floating values corresponding to the data values represented by the scales. The labels displayed along the abscissa scale are the indexes from 0 to 6 referencing the days of the week. To make these values more understandable, we can display the names of the days instead of the indexes. To do this, use the following code:

```
const char* labels[7] = {"Monday", "Tuesday", "Wednesday",
                        "Thursday", "Friday", "Saturday",
                        "Sunday"};
IUInt labelsCount = 7;
abscissaScale->setStepLabels(labelsCount, labels);
```

Adding a Label at the End of the Ordinate Scale Axis

The ordinate scale represents the temperatures expressed in degrees Celsius. To indicate the unit of the values represented by this scale, we can add the label "Celsius" at the end of the axis. To do this, use the following code:

```
ordinateScale->setAxisLabel("Celsius");
```

Adding an Arrow at the End of the Scale Axes

To add an arrow at the end of each scale axis, use the following code:

```
abscissaScale->setAxisOriented(11True);
ordinateScale->setAxisOriented(11True);
```

Figure 5.12 and Figure 5.13 now show the new charts that we obtain. The names of the days of the week appear at the major tick marks along the abscissa. The label "Celsius" appears at the end of the ordinate scale and arrows appear at the end of each axis.

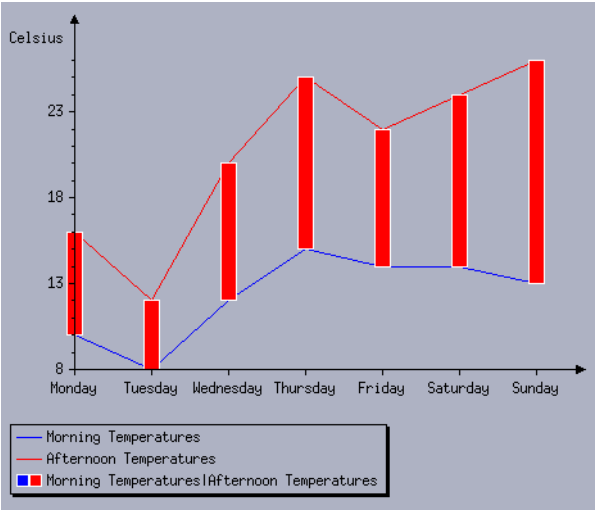


Figure 5.12 The Temperatures Cartesian Chart with Additional Customization

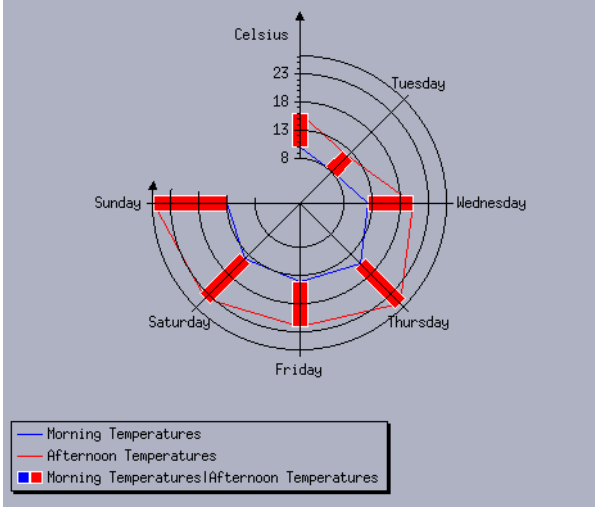


Figure 5.13 The Temperatures Polar Charts with Additional Customization

Note: In step 3 of the section *Customizing the Abscissa Scale* on page 111, we added an instruction for the polar chart to display the step label of the abscissa scale at the intersection point with the ordinate scale. This instruction has been removed to obtain the chart in Figure 5.11. If you keep this instruction, the “Monday” label will be drawn and it will be partially covered by the ordinate axis.

The Charts package of IBM ILOG Views provides many ways for you to customize your charts. For example, you can also add grids, cursors, and so on. For more details about all the possible ways to customize a chart, take a look at the remaining chapters of this user’s manual.

How Charts Work in IBM ILOG Views

A chart object needs several different kinds of objects to display a chart. All these different objects work together in the process of converting data into a graphical display.

Components of a Chart Object

Through the examples in the section *Basic Steps for Creating a Chart* on page 102, you saw that a chart object uses:

- ◆ A chart data object to manage the data sets it displays.
- ◆ Displayer objects to display the graphical representations of the data (polylines, high-low bars, and so on).
- ◆ An optional legend object to represent the legend of the chart.

In addition, a chart object uses the following objects:

- ◆ *Scale displayer* objects to display the scales.
- ◆ A *layout* object to compute the layout of the chart. More specifically, this object computes the position of the area where the data are displayed within the bounding box of the chart object.
- ◆ A *projector* object to perform the projection of data into screen coordinates.
- ◆ A *coordinate information* object to store specific information for each coordinate of the data that is represented by a scale in the chart object.

To see how all these various components work together within a chart object, let’s take a look at the Temperatures chart that we created as an example in *Creating a Simple Cartesian Chart* on page 103 (Figure 5.14).

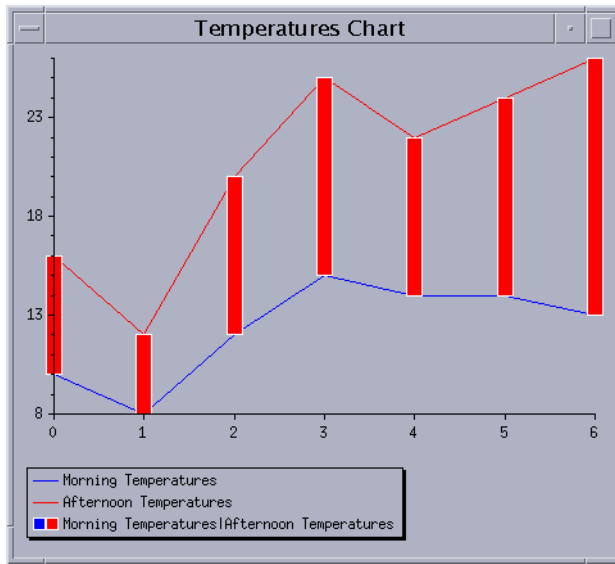


Figure 5.14 Example of the Temperatures Chart

Figure 5.15 shows the objects used by the chart object to display the Temperatures Chart.

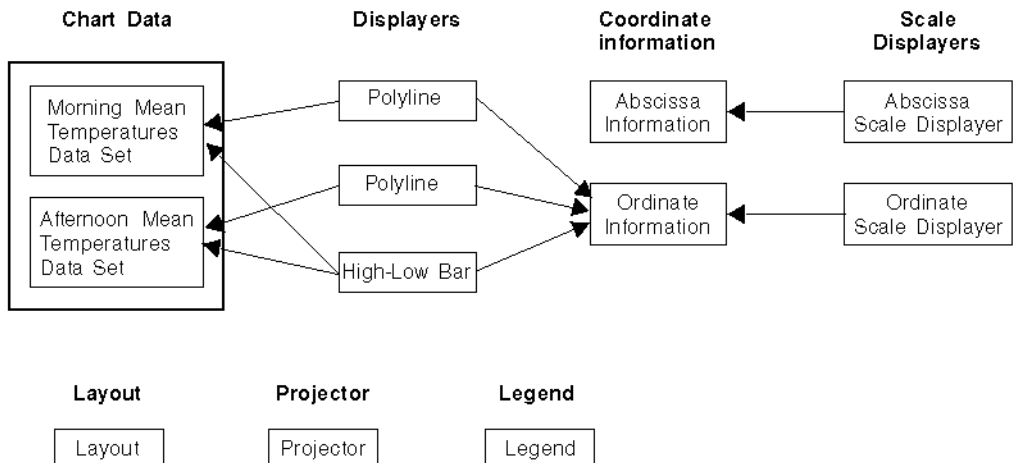


Figure 5.15 Components of the Temperatures Chart

The chart object uses the following objects to display the Temperatures Chart:

- ◆ A chart data object that manages the two data sets: the morning mean temperatures and the afternoon mean temperatures data sets.
- ◆ Three displayer objects: two polyline displayers and one high-low bar displayer.
- ◆ Two coordinate information objects: one associated with the coordinate of the data represented by the abscissa scale and one associated with the coordinate of the data represented by the ordinate scale.
- ◆ Two scale displayer objects: one to display the abscissa scale and one to display the ordinate scale.
- ◆ A layout object that computes the layout of the chart.
- ◆ A projector object that transforms the data points into screen coordinates.
- ◆ A legend object that is used to display information describing the data displayed by the chart.

The arrows in the figure show the relationship between some of these objects. For each of the displayers (the two polylines and the high-low bar), the data set(s) it displays is indicated as well as the coordinate information object associated with the ordinate scale considered by the displayer. The coordinate information is necessary since a chart can use several ordinate scales (this means that several coordinate systems can be displayed on the same chart) and a displayer object must know which coordinate system it must use to display the data. For each of the scale displayers, the arrows indicate the coordinate information object that is associated with the coordinate represented by the scale displayer.

Component Classes of the Charts Library

A chart object is an instance of the `IlvChartGraphic` class or one of its derived classes in the Charts Library. (See *Chart Classes* on page 101 for more details.)

Each component object used by a chart object to convert data into a graphical display corresponds to a dedicated object in the Charts Library as well.

Chart Data and Data Set Objects

A chart data object is an instance of a subclass of the `IlvAbstractChartData` class. The data sets managed by the chart data object are instances of subclasses of the `IlvChartDataSet` class. (See *Data Classes* on page 100 for more details.)

Default Instantiation

When you create a chart object, an `IlvMemoryChartData` object is created by default as the chart data object used to manage the data sets for the chart.

Displayer Objects

Displayer objects are instances of subclasses of the `IlvAbstractChartDisplayer` class. They are used to draw the graphical representations of the data.

Default Instantiation

No displayer objects are created by default when you create a chart object. You have to create the displayers that you want to use to display your data and set them on the created chart.

A displayer object stores:

- ◆ Pointers to the data sets it displays.
- ◆ A pointer to the coordinate information associated with the ordinate scale it considers to display the data.

Scale Displayer Objects

Scale displayer objects are instances of subclasses of the `IlvAbstractScaleDisplayer` class. They are used to draw the scales of a chart. The computation of the steps and substeps of a scale is performed by means of a *scale steps updater* object that is set on the scale. This object is an instance of a subclass of the `IlvScaleStepsUpdater` class.

Default Instantiation

No scale displayer objects are created by default in an `IlvChartGraphic` object since the type of the scales depends on the type of the object used to project the data into screen coordinates.

However, when you create:

- ◆ an `IlvCartesianChart` object, two `IlvRectangularScaleDisplayer` objects are created by default: one to display the rectangular abscissa scale and one to display the rectangular ordinate scale.
- ◆ an `IlvPolarChart` object, an `IlvCircularScaleDisplayer` object is created by default to display the circular abscissa scale and an `IlvRectangularScaleDisplayer` object is created by default to display the rectangular ordinate scale.

When a scale displayer object is created, an *automatic* scale steps updater object (instance of the `IlvAutoScaleStepsUpdater` class) is created by default and set on the scale.

Layout Object

The layout object must be an instance of the `IlvChartLayout` class or one of its derived classes. The layout object handles the computation of the chart layout and the computation of the data display area within the bounding box of the chart.

Default Instantiation

When you create a chart object, an `IlvChartLayout` object is created by default to compute the layout.

Projector Object

A projector object is an instance of a subclass of the `IlvAbstractProjector` class. It is used to project the data into screen coordinates.

Default Instantiation

No projector object is created by default in an `IlvChartGraphic` object since the type of projector used to project the data depends on the type of the coordinate system in which the displayed data are expressed.

However, when you create:

- ◆ an `IlvCartesianChart` object, an `IlvCartesianProjector` object that projects data expressed in a Cartesian system of coordinates is created by default.
- ◆ an `IlvPolarChart` object, an `IlvPolarProjector` object that projects data expressed in a polar system of coordinates is created by default.

Coordinate Information Objects

The coordinate information objects must be instances of the `IlvCoordinateInfo` class or one of its derived classes. They are used to store specific information for each coordinate of the data that is represented by a scale.

Default Instantiation

No coordinate information objects are created by default in an `IlvChartGraphic` object because the scale displays are not created at that level and a coordinate information object must be created for each coordinate that is represented by a scale.

However, two `IlvCoordinateInfo` objects are created by default in the `IlvCartesianChart` and `IlvPolarChart` objects: one associated with the abscissa scale and one associated with the ordinate scale that are created by default.

A coordinate information object stores:

- ◆ The minimum and maximum values considered for the coordinate with which the coordinate information object is associated. These values are used to select the data to display and to specify the minimum and maximum values for the scale that represents this coordinate.
- ◆ Optionally, a *transformer* object that specifies the transformation that will be applied to the coordinate. The transformer object must be an instance of a subclass of the `IlvCoordinateTransformer` class.

Legend Object

A legend object must be an instance of the `IlvChartLegend` class or one of its derived classes. It is used to display the legend corresponding to the chart displayed.

Default Instantiation

No legend object is created by default when you create a chart object. You have to create the legend object you want to be used and to set it on the created chart. (For information on how to create and set a legend object, see *Adding the Legend* on page 112.)

Using the Component Classes in an IlvChartGraphic Object

To see how the component objects are used in a chart object, let's take another look at our Temperatures Chart (see Figure 5.14). Instead of constructing this chart from an `IlvCartesianChart` object (see *Creating a Simple Cartesian Chart* on page 103), we are going to construct it from an `IlvChartGraphic` object, the base object that allows us to display a chart.

Figure 5.15 shows the component objects that are needed to display this chart. When we created the chart using an `IlvCartesianChart` object, we set the following objects by hand:

- ◆ The data sets
- ◆ The displayers
- ◆ The legend

If we use an `IlvChartGraphic` object instead of an `IlvCartesianChart` object, we must set the following objects as well:

- ◆ The projector object
- ◆ The coordinate information objects
- ◆ The scale displayer objects

The layout object does not need to be set since an instance is created by default when you create an `IlvChartGraphic` object.

To create our Temperatures Chart from an `IlvChartGraphic` object, we will again follow the basic steps presented in *Creating a Simple Cartesian Chart* on page 103. However, we must replace the step *Creating a Cartesian Chart* with the following series of tasks:

- ◆ *Creating a Chart Graphic Object*
- ◆ *Creating and Setting the Projector*
- ◆ *Creating the Coordinate Information for the Abscissa*
- ◆ *Creating and Setting the Abscissa Scale Displayer*
- ◆ *Creating the Coordinate Information for the Ordinate*
- ◆ *Creating and Setting the Ordinate Scale Displayer*

All the other tasks that are described in the paragraph *Creating a Simple Cartesian Chart* on page 103 remain the same.

The complete source code of the example can be found in the `chartgraphic.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory.

Creating a Chart Graphic Object

To create a chart graphic object, use the following code:

```
IlvChartGraphic* chart = new IlvChartGraphic(display,  
                                             IlvRect(10, 10, 450, 300));
```

The `IlvRect` object passed as a parameter corresponds to the bounding box of the created chart.

Creating and Setting the Projector

Since the data to be displayed with our chart are expressed in Cartesian coordinates, the projector that is set is an instance of the `IlvCartesianProjector` class.

```
chart->setProjector(new  
IlvCartesianProjector(IlvCartesianProjector::IlvXRightYTop));
```

We indicate the orientation of the scales as a parameter: the abscissa scale is oriented toward the right of the screen and the ordinate scale is oriented toward the top. All orientations that keep the abscissa and the ordinate scale(s) orthogonal are possible.

Creating the Coordinate Information for the Abscissa

Use the following code to create the coordinate information for the abscissa:

```
IlvCoordinateInfo* abscissaCoordInfo  
    = new IlvCoordinateInfo(IlvAbscissaCoordinate);
```

The created coordinate information is associated with a coordinate. The type of this coordinate is passed as a parameter to the constructor used to create this coordinate information. By default, the minimum and maximum values that will be considered for the scale that represents this coordinate are automatically computed from the abscissa values of the data that have to be displayed by the created chart. This is done so that all the defined data can be displayed.

Creating and Setting the Abscissa Scale Displayer

To define the abscissa scale displayer, perform the following steps:

1. Create the displayer for the abscissa scale.

Since the data to be displayed with our chart are expressed in Cartesian coordinates, the object that is set to display the scale representing the abscissa values is an instance of the `IlvRectangularScaleDisplayer` class.

```
IlvRectangularScaleDisplayer* abscissaScale
    = new IlvRectangularScaleDisplayer(abscissaCoordInfo,
                                       chart->getPalette());
```

The coordinate information object associated with the coordinate that is represented by the abscissa scale is passed as a parameter to the constructor that creates the displayer for the abscissa scale. The palette that will be used to display the abscissa scale is set to the default palette of the created chart.

2. Set the displayer for the abscissa scale.

```
chart->setAbscissaScale(abscissaScale);
```

3. Customize the abscissa scale.

The format used for the values that are displayed next to the main graduations of a scale is set by default to `"%.2f"`. Since the abscissa scale is used to represent integer values corresponding to the indexes of the days of the week (from 0 to 6), the format must be set to `"%.0f"` to avoid having the indexes of the days represented by floating values.

```
abscissaScale->setStepLabelFormat("%.0f");
```

The flag indicating whether the step labels must be drawn where the axes intersect is set by default to `IlvFalse` for a scale. Thus the label "0" for the first day of the week will not be drawn since the corresponding graduation is located on the abscissa scale at its intersection with the ordinate scale. To make this label appear, the flag must be set to `IlvTrue`.

```
abscissaScale->drawLabelOnCrossings(IlvTrue);
```

Creating the Coordinate Information for the Ordinate

To create the coordinate information for the ordinate, use the following code:

```
IlvCoordinateInfo* ordinateCoordInfo
    = new IlvCoordinateInfo(IlvOrdinateCoordinate);
```

The created coordinate information is associated with a coordinate. The type of this coordinate is passed as a parameter to the constructor used to create this coordinate information. By default, the minimum and maximum values that will be considered for the scale that represents this coordinate are automatically computed from the ordinate values of the data that have to be displayed along this scale. This is done so that all the defined data that have to be displayed along this scale can be displayed.

Creating and Setting the Ordinate Scale Displayer

To define the ordinate scale displayer, perform the following steps:

1. Create the displayer for the ordinate scale.

```
IlvRectangularScaleDisplayer* ordinateScale
    = new IlvRectangularScaleDisplayer(ordinateCoordInfo,
                                       chart->getPalette());
```

Just as with the abscissa scale, the coordinate information object associated with the coordinate that is represented by the ordinate scale is passed as a parameter to the constructor that creates the displayer for the ordinate scale. The palette that will be used to display the ordinate scale is set to the default palette of the created chart.

2. Set the displayer for the ordinate scale.

```
chart->addOrdinateScale(ordinateScale);
```

3. Customize the ordinate scale.

The flag indicating whether the step labels must be drawn where the axes intersect is set by default to `IlvFalse` for a scale. Thus the label "8" for the minimum temperature value that is displayed will not be drawn since the corresponding graduation is located on the ordinate scale at its intersection with the abscissa scale. To make this label appear, the flag must be set to `IlvTrue`.

```
ordinateScale->drawLabelOnCrossings(IlvTrue);
```

At this point, we have now completed creating our Cartesian chart from an `IlvChartGraphic` object. The chart we obtain is the same as the one shown in Figure 5.7. You can readily see the difference in using an `IlvChartGraphic` object instead of using an `IlvCartesianChart` object. Many more lines of code were required to instantiate the same chart object that we obtained by using a single line of code:

```
IlvChartGraphic* chart = new IlvCartesianChart(display,
                                              IlvRect(10, 10, 450, 300));
```

How Displayer Objects Draw the Graphical Display

The drawing of the graphical representations of data is performed by the displayer objects. In order to draw a graphical representation of the data, a displayer needs to have the data points of the data set(s) it is going to display expressed in screen coordinates.

Transforming Data Points into Screen Coordinates

Figure 5.16 illustrates how the data points represented by a given data set are transformed into screen coordinates.

1. From a given data set, we get the data points in the form $P(C_0, C_1)$ that are represented by the data set. C_0 is the abscissa value and C_1 the ordinate value of the data point.
2. Each data point P is transformed into P' by applying the transformer defined for the abscissa coordinate to its abscissa value C_0 and the transformer defined for the ordinate coordinate to its ordinate value C_1 . These transformers are stored in the coordinate

information objects associated with the abscissa and the ordinate scales considered by the displayer to display the data set. The same abscissa coordinate information object is used by all the displayers since the chart uses only one abscissa scale. The ordinate coordinate information object that must be used by a given displayer is stored in this displayer (in the form of a pointer).

3. The transformed point P' is passed to the projector that finally projects the point into the point $p(x, y)$ expressed in screen coordinates. (For details about this step, see *Projecting Transformed Data Points into Screen Coordinates* on page 124.)

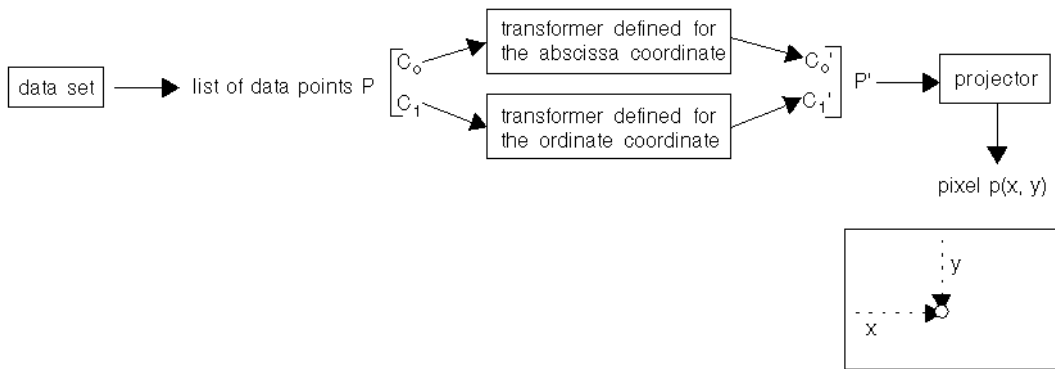


Figure 5.16 *Transforming Data Points into Screen Coordinates*

Projecting Transformed Data Points into Screen Coordinates

To project the transformed data points into screen coordinates (see step 3 on page 124), the projector needs the following information:

- ◆ The area on the screen where the data points are to be displayed. This is called the *data display area* and is specified by the layout object.
- ◆ The minimum and maximum data values represented by the abscissa and the ordinate scales that are considered by the displayer to display the data. The minimum and maximum data values are the lower and upper bounds of the data points that will be projected into screen coordinates. These values are stored in the coordinate information objects associated with the abscissa and ordinate scales.

Using this information, the projector then maps the transformed data points into screen coordinates so that the data points occupy the largest amount of space within the data display area.

Figure 5.17 illustrates these concepts. (Once again, we are using our Temperatures Chart as an example.) Our chart uses one abscissa scale and one ordinate scale. Therefore, all of the displayers consider the same ordinate scale when displaying the data. By default, the

minimum and maximum data values that are considered for the abscissa and ordinate scales of the chart are automatically computed so that all the data points can be displayed. For the abscissa scale, these values correspond to the minimum and maximum abscissa values in the lists of data points of the morning and afternoon mean temperatures data sets. For the ordinate scale, these values correspond to the minimum and maximum ordinate values in the lists of data points of the morning and afternoon mean temperatures data sets.

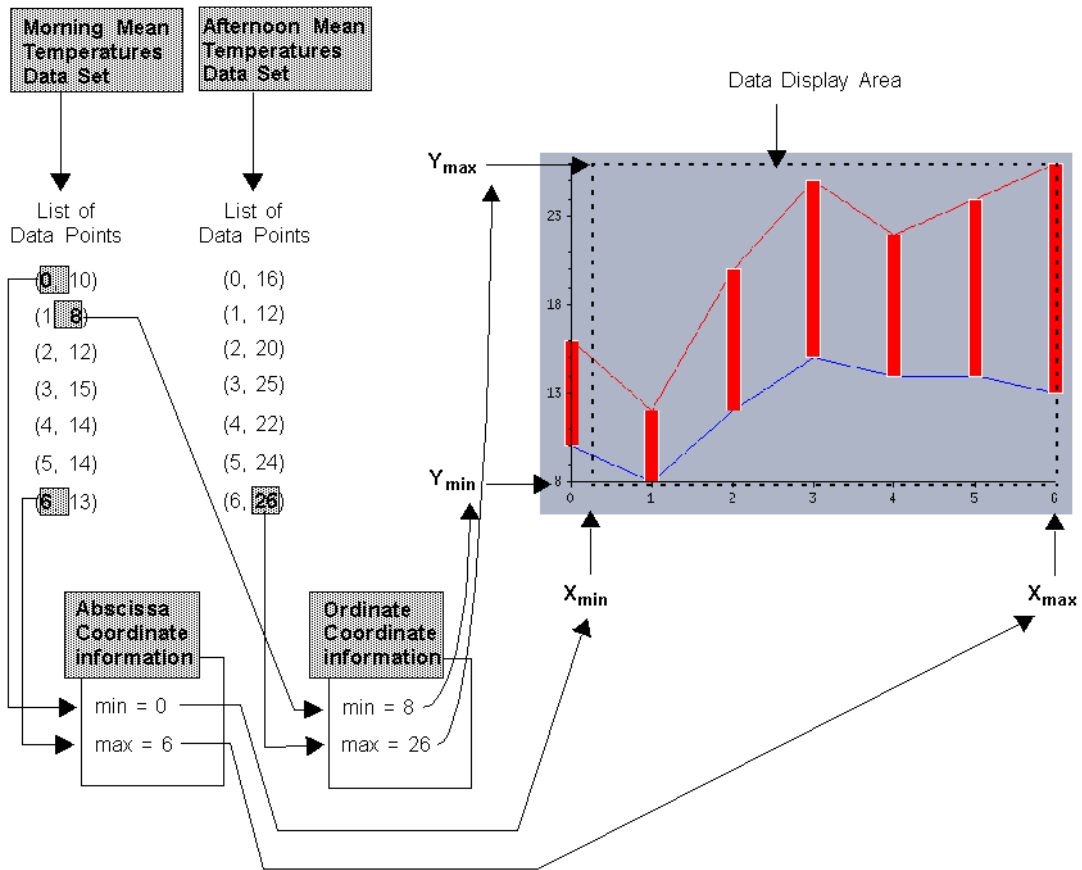


Figure 5.17 Mapping the Temperatures Data into Screen Coordinates

Data Handling

This chapter provides detailed information on data handling for the charts. You will find information on the following topics:

- ◆ *Handling Data Storage*
- ◆ *Sharing Data Among Charts*
- ◆ *Modifying Data and Updating Charts*

Handling Data Storage

The Charts Library has been designed with a clear separation between the data to be displayed and the graphical representation of the data. The storage of the data sets that have to be displayed by a given chart object is handled by a dedicated object referred to as the *chart data* object.

The base class used to represent a chart data object is the `IlvAbstractChartData` class. A subclass called `IlvMemoryChartData` class is provided in the library. The `IlvMemoryChartData` class stores the data sets it manages in memory. When you create a chart object, an instance of the `IlvMemoryChartData` class is created by default and is set on the created chart object to handle the data sets it is going to display.

The data sets managed by a chart data object are all instances of subclasses of the `IlvChartDataSet` class. The subclasses of the `IlvChartDataSet` class that are provided in the library define the different types of data sets that can be displayed in a chart.

Types of Data Sets

A data set to be displayed by a given chart can be defined as:

- ◆ A set of points with two coordinates (set-of-points data set)
- ◆ A set of values (set-of-values data set)
- ◆ A function of the type $y = f(x)$ (function data set)
- ◆ A cyclic set of points (cyclic set-of-points data set)

Set-of-Points Data Set

A set-of-points data set is represented by an instance of the `IlvChartPointSet` class. The data points managed by this data set are instances of the `IlvDoublePoint` class. The `IlvDoublePoint` class allows you to represent a point with two coordinates of the `IlvDouble` type.

The following table shows an example of some data that can be represented by an `IlvChartPointSet` object. Each point (x, y) is stored by using an `IlvDoublePoint` instance within the `IlvChartPointSet` object.

X	Y
0.5	1.0
1.2	2.3
1.6	3.1

This data set can be created by using the following code:

```
IlvChartPointSet* dataSet = new IlvChartPointSet();
dataSet->addPoint(IlvDoublePoint(0.5, 1.0));
dataSet->addPoint(IlvDoublePoint(1.2, 2.3));
dataSet->addPoint(IlvDoublePoint(1.6, 3.1));
```

Set-of-Values Data Set

A set-of-values data set is represented by an instance of the `IlvChartYValueSet` class. The data values managed by this data set are of the `IlvDouble` type. The values correspond to the ordinates of the data points that will be displayed. The abscissas of the data points are by definition the indexes of the stored values.

The following table shows an example of data that can be represented by an `IlvChartYValueSet` object.

X	Y
0	1.2
1	3.1
2	4.6

Each y-value is stored using an `IlvDouble` instance within the `IlvChartYValueSet` object. The x-values are not stored because they correspond to the indexes of the stored y-values.

A set-of-values data set can be created by using the following code:

```
IlvChartYValueSet* dataSet = new IlvChartYValueSet();
dataSet->addValue(1.2);
dataSet->addValue(3.1);
dataSet->addValue(4.6);
```

This code is equivalent to the following:

```
IlvChartYValueSet* dataSet = new IlvChartYValueSet();
dataSet->addPoint(IlvDoublePoint(0.0, 1.2));
dataSet->addPoint(IlvDoublePoint(1.0, 3.1));
dataSet->addPoint(IlvDoublePoint(2.0, 4.6));
```

Function Data Set

A function data set is defined with a function of the type $y=f(x)$. It is represented by an instance of a subclass of the `IlvAbstractChartFunction` class.

The abscissa values of the data points are computed from the minimum and maximum values considered for the abscissa and from the number of data points. The ordinate values of the data points are then obtained by applying the function to the computed abscissa values. The abscissa and the ordinate values of the i^{th} data point that is represented by a data set defined with a function f are obtained by using the following formula:

$$x = ((_xMax - _xMin) / (_dataCount - 1)) * i + _xMin;$$

$$y = f(x);$$

where `_xMin` and `_xMax` are the minimum and the maximum values considered for the abscissa and `_dataCount` is the number of considered data points. The values `_xMin`, `_xMax`, `_dataCount`, and the function f have to be defined by the user.

Two subclasses of the `IlvAbstractChartFunction` class are available in the library:

- ◆ `IlvCallbackChartFunction` for which the function to be represented is defined by a callback. This callback must be of the `IlvDoubleFunction` type:

```
typedef IlvDouble (* IlvDoubleFunction)(IlvDouble);
```


- ◆ `IlvScriptChartFunction` for which the function to be represented is defined by a script.

Either one of these two subclasses can be used to represent any function of the type $y = f(x)$. The following examples show how to use each of the subclasses. We will create an `IlvCallbackChartFunction` object and an `IlvScriptChartFunction` object, each representing the square function $y = x^2$. Both data sets are initialized with 0 and 5 as the minimum and maximum values for the abscissa and 9 as the number of considered data points.

Using an `IlvCallbackChartFunction` Data Set Object

To create an `IlvCallbackChartFunction` object, perform the following steps:

1. Define the callback function for computing the square function.

```
IlvDouble
square(IlvDouble x)
{
    return x*x;
}
```

2. Create the `IlvCallbackChartFunction` data set representing the square function.

```
IlvCallbackChartFunction* function =
    new IlvCallbackChartFunction(IlvCoordInterval(0.,5.), 9, square);
```

The minimum and maximum values considered for the abscissa are set to 0 and 5, respectively. The number of data points considered is set to 9 and the callback function is set to square.

Using an `IlvScriptChartFunction` Data Set Object

This example shows you how to create an `IlvScriptChartFunction` object directly in your code. However, you can easily create an `IlvScriptChartFunction` data set by using the Chart Inspector in IBM ILOG Views Studio (see *Using the Chart Inspector* on page 20).

To create an `IlvScriptChartFunction` object, perform the following steps:

1. Set a script context on the holder of the container or manager that contains the chart, if a script context is not defined for the holder.

The script function used in an `IlvScriptChartFunction` object is assumed to be stored in a script context associated with the holder of the container or manager that contains the chart. If a script context is not defined for the holder, a script context must be created and set on the holder. To do this, use the following code:

First, create a script context using the JavaScript™ scripting language:

```
const IlvSymbol* scriptLanguageName = IlvGetSymbol("JvScript");
IlvScriptLanguage* javascript = IlvScriptLanguage::Get(scriptLanguageName);
IlvJvScriptContext* javascriptContext = new IlvJvScriptContext(javascript, 0);
```

Then, set the created script context on the holder of the container or manager that contains the chart:

```
cont->getHolder()->setScriptContext(javascriptContext);
```

2. Load the script function for computing the square function into the created script context.

```
IlvScriptContext* context =
    (cont->getHolder()->getScriptContext(scriptLanguageName));
context->loadScript("../data/scriptfile");
```

Since several script contexts may be associated with the holder, the name of the scripting language (JavaScript in this example) is passed as a parameter to the `getScriptContext` method in order to retrieve the script context using this scripting language. The file `scriptfile` contains the implementation of the script function in JavaScript:

```
function square(x)
{
    return x*x;
}
```

3. Create the `IlvScriptChartFunction` data set representing the square function.

```
IlvScriptChartFunction* function =
    new IlvScriptChartFunction(IlvCoordInterval(0.,5.),
                              9,
                              "square",
                              cont->getHolder(),
                              scriptLanguageName);
```

The minimum and maximum values considered for the abscissa are set to 0 and 5, respectively. The number of data points considered is set to 9. The name of the script function is set to "square". The holder in which the script function will be retrieved is set to the holder of the container or manager that contains the chart. The name of the scripting language used to write the script function is set to `scriptLanguageName`.

Cyclic Set-of-Points Data Set

A cyclic set-of-points data set is represented by an instance of the `IlvChartCyclicPointSet`. It is similar to the normal set-of-points data set described above, except that when the number of added points reaches a fixed maximum number, the "old" points are discarded so that the actual number of points kept in memory remains constant. As shown by the chart, the count will grow until it reaches the limit and then it remains constant. Each time a new point is added, it will be at the last index, while the old ones are shifted back by one.

The maximum number of points can be set with `setMaxCount()` and is infinite by default.

For example:

```

IlvChartDataSet* dataSet;
dataSet = new IlvChartCyclicPointSet("Cyclic point set");
dataSet->setMaxCount(100);

```

For an example of how to use `IlvChartCyclicPointSet`, see in the `shiftcar` sample in `samples/charts/scrolling` directory.

Adding Data Sets to Be Displayed by a Chart

To display a data set within a given chart, the data set must be added to the chart data object that is set on the chart. You can do this in two ways:

- ◆ Set the data sets for the chart data object that is created by default and set on the chart.
- ◆ Create your own chart data object, set the data sets for this chart data object, and then set the chart data object for the chart.

Examples

In the following examples, we will show you how to add two data sets, an `IlvChartPointSet` data set and an `IlvChartYValueSet` data set, to a chart object.

```

IlvChartDataSet* dataSets[2];
//== Creating the IlvChartPointSet data set and fill it with data.
dataSets[0] = new IlvChartPointSet();
dataSets[0]->addPoint(IlvDoublePoint(0.5, 1.0));
dataSets[0]->addPoint(IlvDoublePoint(1.2, 2.3));
dataSets[0]->addPoint(IlvDoublePoint(1.6, 3.1));

//== Creating the IlvChartYValueSet data set and fill it with data.
dataSets[1] = new IlvChartYValueSet();
dataSets[1]->addPoint(IlvDoublePoint(0, 1.2));
dataSets[1]->addPoint(IlvDoublePoint(1, 3.1));
dataSets[1]->addPoint(IlvDoublePoint(2, 4.6));

```

The first example will show you how to add the data sets to the default chart data object. The second example will show you how to add the data sets to your own chart data object.

Adding the Data Sets to the Default Chart Data Object

The chart data object that is set on a given chart object to manage the data sets it displays is obtained by using the `IlvChartGraphic::getData` method.

You can use the following code to add the data sets directly to the chart data object that has been created by default and set for the chart object:

```

chart->getData()->addDataSet(dataSets[0]);
chart->getData()->addDataSet(dataSets[1]);

```

Or more simply, you can add the two data sets at the same time:

```

IUUInt dataSetsCount = 2;
chart->getData()->setDataSets(dataSetsCount, dataSets);

```

Adding Data Sets to Your Own Chart Data Object

To use your own chart data object to manage the data sets to be displayed by a given chart object, do the following:

1. Set the data sets directly for your chart data object.

```
IlvAbstractChartData* myChartData = new IlvMemoryChartData();  
myChartData->addDataSet(dataSets[0]);  
myChartData->addDataSet(dataSets[1]);
```

2. Set this chart data object for the chart object.

```
chart->setData(myChartData);
```

Note: You can use whatever chart data object you want as long as it is a derived object of the `IlvAbstractChartData` class.

Sharing Data Among Charts

With the Charts Library, data can be shared among different charts. The same data set can be set for several chart data objects and the same chart data object can be set on several chart objects. Figure 6.1 illustrates these basic rules.

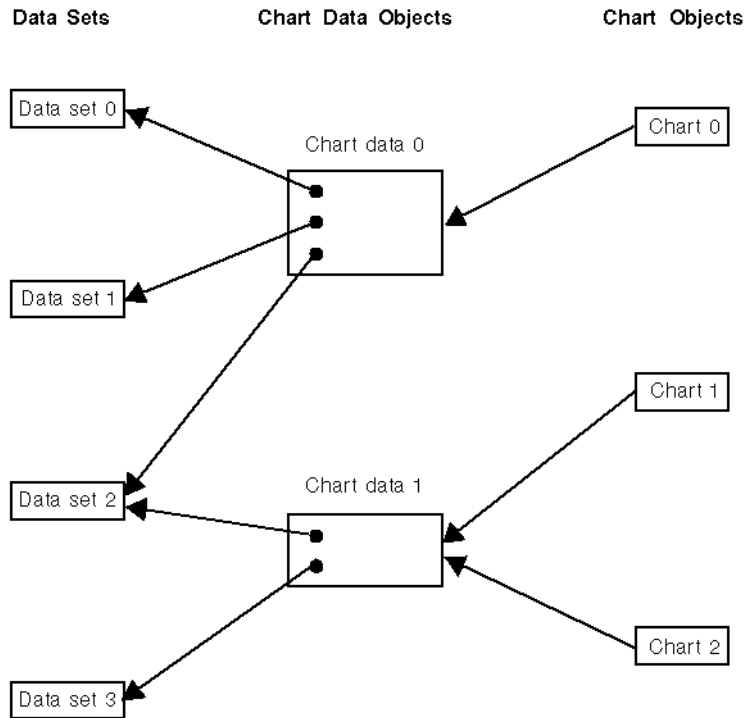


Figure 6.1 *Sharing Data*

In this figure, *Data set 2* is used by both the *Chart data 0* and the *Chart data 1* chart data objects. The chart data object *Chart data 1* is set on both the *Chart 1* and the *Chart 2* chart objects to manage the data sets they display. This means that *Chart 1* and *Chart 2* will display the same data.

Data sharing is possible due to a *lock/unlock* system that has been implemented for data sets and for chart data objects. Each time you add a data set to a chart data object, the data set will be automatically locked by a call to the `IlvChartDataSet::lock` method. In the same way, each time you set a chart data object on a chart object, the chart data object will be automatically locked by a call to the `IlvAbstractChartData::lock` method.

Similarly, each time you remove a data set from a chart data object, the data set will be automatically unlocked by a call to the `IlvChartDataSet::unLock` method. Each time you remove a chart data object from a chart object, the chart data object will be automatically unlocked by a call to the `IlvAbstractChartData::unLock` method.

(See the `lock` and `unLock` methods of the `IlvChartDataSet` and `IlvAbstractChartData` class in the reference manual for more information.)

Modifying Data and Updating Charts

The charts created in IBM® ILOG® Views are *data-aware*. Data can be changed “on the fly” and all charts that display the data will be automatically updated to reflect the changes. The charts are automatically updated using *listeners* that are set on the data. The listeners catch data changes and can then be used to perform specific tasks according to the types of modifications that are made to the data.

Types of Modifications

The data displayed in a chart can be modified in different ways depending on whether the change is made at the level of the data set or at the level of the chart data object.

Modifications Made at the Data Set Level

The following types of modifications can be made to a data set:

- ◆ A new data item can be added to the data set.

This can be done by using one of the following methods:

```
IlvChartDataSet::setPoint
```

```
IlvChartDataSet::addPoint
```

```
IlvChartDataSet::insertPoint
```

- ◆ An existing data item in the data set can be replaced by another data item.

This can be done by using the `IlvChartDataSet::setPoint` method. This method sets the data item stored at a given index in a data set. If a data item already exists at the index, it is replaced by the new one.

- ◆ A data item can be removed from the data set.

This can be done by using the `IlvChartDataSet::removePointAndInfo` method. This method also removes the point information object (if any) that is associated with the data item.

Note: A method is also available to remove all the data items from a given data set at the same time: `IlvChartDataSet::removePointsAndInfo`.

Modifications Made at the Chart Data Object Level

The following types of modifications can be made to a chart data object:

- ◆ A new data set can be added to a chart data object.

This can be done by using one of the following methods:

```
IlvAbstractChartData::setDataSet
```

```
IlvAbstractChartData::addDataSet
```

```
IlvAbstractChartData::insertDataSet
```

Note: A method is also available to set several data sets on a given chart data object at the same time: `IlvAbstractChartData::setDataSets`.

- ◆ An existing data set in a chart data object can be replaced by another data set.

This can be done by using one of the following methods:

`IlvAbstractChartData::setDataSet` sets the data set stored at a given index in a chart data object. If a data set already exists at this index, it is replaced by the new one.

`IlvAbstractChartData::replaceDataSet` replaces a given data set by a new one.

- ◆ A data set can be removed from a chart data object.

This can be done by using the `IlvAbstractChartData::removeDataSet` method.

Note: A method is also available to remove all the data sets from a given chart data object at the same time: `IlvAbstractChartData::removeDataSets`.

Updating Charts Automatically

Charts are automatically updated when modifications are made to the data they display. This automatic update is performed by *listeners* that are set on the data.

Notification Mechanism Based on Listeners

A mechanism based on the use of listeners has been implemented to propagate automatically all modifications made to the data to the objects using these data. A listener is a dedicated object that catches the modifications made to the object on which the listener is set and then notifies other objects about these modifications.

Modifications can be made to data at the level of the data sets and at the level of the chart data objects (see *Types of Modifications* on page 134). Therefore, listeners are also set at the level of the data sets and at the level of the chart data objects. Figure 6.2 illustrates how listeners are set at the data set level and the chart data object level.

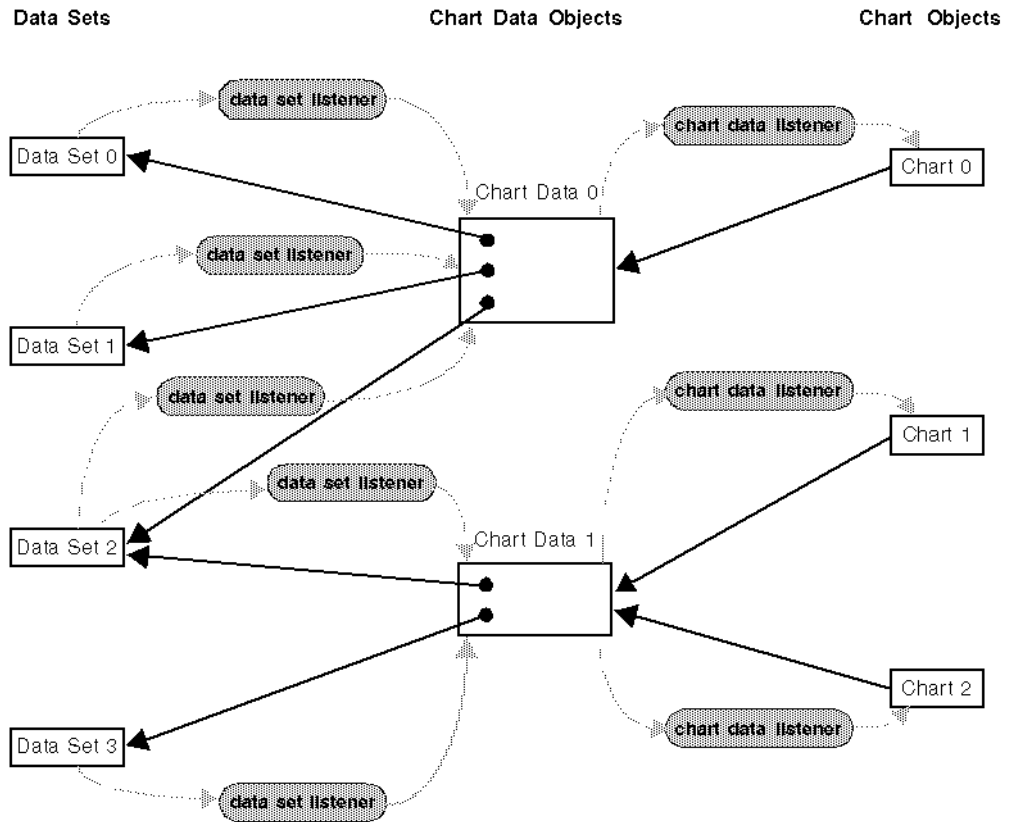


Figure 6.2 Listeners at the Data-Set Level and at the Chart-Data-Object Level

Listeners are set on each data set to notify the chart data objects that use the data set about the modifications made to the data set. Notice that two listeners are set on the data set *Data set 2* since this data set is used by two chart data objects, *Chart data 0* and *Chart data 1*. These listeners are automatically set on the data sets when the data sets are set on the chart data objects.

In the same way, listeners are set on each chart data object to notify the chart objects that use the chart data object about the modifications made to the chart data object. Notice that two listeners are set on the chart data object *Chart data 1* since this chart data object is used by two chart objects, *Chart 1* and *Chart 2*. These listeners are automatically set on the chart data objects when the chart data objects are set on the chart objects.

How the Update is Performed

By default, the listeners are enabled for all data sets. This means that listeners will be used to propagate the modifications made to the data sets. If the listeners are not enabled, they will not be considered and thus the modifications will not be transmitted to the listeners.

The method `IlvChartDataSet::areListenersEnabled` indicates whether the listeners are enabled for a given data set. The method `IlvChartDataSet::enableListeners` is used to specify whether the listeners should be enabled for a given data set.

Figure 6.3 shows how the listeners propagate a modification made to a data set to all the objects using the data set, thus allowing these objects to be updated. This figure uses the data set *Data set 2* from Figure 6.2 as an example.

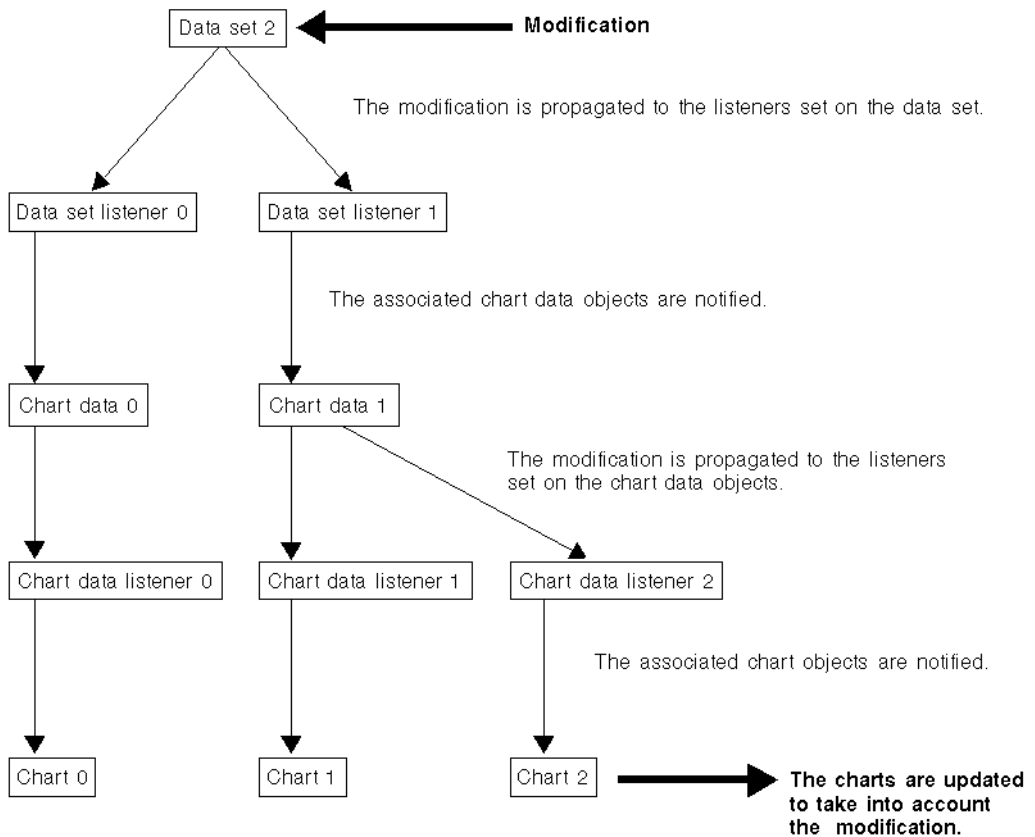


Figure 6.3 Propagation of a Modification Made to a Data Set

If the listeners are enabled, the following events occur as soon as a modification is made to a data set:

1. First, the modification is propagated to all the listeners set on the data set.
2. Each data set listener then notifies each chart data object with which it is associated about the modification.
3. The modification is then propagated to all the listeners set on the chart data objects.
4. Each chart data listener then notifies each chart object with which it is associated about the modification.
5. The charts are finally updated to reflect the modification.

If the listeners are not enabled, the updates needed because of a modification made to a data set will not be performed automatically since the modifications will not be transmitted to the listeners. The updates will have to be made by hand.

Batching the Modifications

By default, the notification of a modification and the update to the chart resulting from the modification is made as soon as the modification occurs. However, if it is not necessary to see the updates one by one as they occur, you can batch modifications when adding new data items to a data set. Using a batch operation when adding new data items increases performance and can be very useful, especially when adding real-time data.

To trigger the batch operations for adding new data items on a given data set, you have to call the `IlvChartDataSet::startBatch` method on the data set. Once the `IlvChartDataSet::startBatch` method has been called, the updates resulting from the addition of new data items are no longer performed. You have to call the `IlvChartDataSet::endBatch` method on the data set to specify that you want to stop the batch operations and perform the global update one time.

The `IlvChartDataSet::startBatch` method can be called several times in succession on a given data set. When this is done, the `IlvChartDataSet::endBatch` method has to be called as many times as the `IlvChartDataSet::startBatch` method. When the last call to the `IlvChartDataSet::endBatch` method is performed, all the data items that have been added since the first call of the `IlvChartDataSet::startBatch` method will be considered all at one time. The update resulting from the addition of all these data items will be performed at one time. The added data items will be drawn all at one time, instead of being drawn one by one as is done when the modifications are not batched.

Using Listeners to Catch Data Changes

The listeners set on data catch modifications made to the data. Listeners are set at the level of the data sets and at the level of the chart data objects. For both types of listeners, a method is defined for each type of modification that can be made to the object on which the listener

is set. The method corresponding to a given modification is called when the modification that has occurred is propagated to the listener (see Figure 6.3).

Listeners on Data Sets

Listeners set on data sets are instances of subclasses of the `IlvChartDataDataSetListener` class. The `IlvChartDataDataSetListener` class is used to watch for all the modifications that can be made to a given data set. If you remember, these modifications include adding new data items, replacing existing data items, and removing data items (see *Modifications Made at the Data Set Level* on page 134). Methods corresponding to these types of modifications are defined in the `IlvChartDataDataSetListener` class:

- ◆ `IlvChartDataDataSetListener::dataPointAdded` is called when a new data item is added to the data set.
- ◆ `IlvChartDataDataSetListener::dataPointChanged` is called when an existing data item is changed in the data set.
- ◆ `IlvChartDataDataSetListener::dataPointRemoved` is called when a data item is removed from the data set.

Listeners on Chart Data Objects

Listeners set on chart data objects are instances of subclasses of the `IlvChartDataListener` class. The `IlvChartDataListener` class is used to listen to all the modifications that can be made to a given chart data object. These modifications include adding new data sets, changing existing data sets, and removing data sets (see *Modifications Made at the Chart Data Object Level* on page 134). Methods corresponding to these types of modifications are defined in the `IlvChartDataListener` class:

- ◆ `IlvChartDataListener::dataSetAdded` is called when a new data set is added.
- ◆ `IlvChartDataListener::dataSetChanged` is called when an existing data set is changed.
- ◆ `IlvChartDataListener::dataSetRemoved` is called when a data set is removed.

Since a chart data object handles data sets, the modifications made at the level of a data set are also reported at the level of the chart data object. The corresponding methods that are defined in the `IlvChartDataListener` class are:

- ◆ `IlvChartDataListener::dataPointAdded` is called when a new data item is added to a data set.
- ◆ `IlvChartDataListener::dataPointChanged` is called when an existing data item is changed in a data set.
- ◆ `IlvChartDataListener::dataPointRemoved` is called when a data point is removed from a data set.

Defining Your Own Listeners

You can define your own data listeners to perform specific tasks when certain types of modifications are made to a data set or to a chart data object. To do so, you need to perform these steps:

1. Create a subclass of the `IlvChartDataSetListener` class (or the `IlvChartDataListener` class).
2. Implement the specific tasks you want to be performed when some types of modifications are made by subclassing the corresponding methods. The methods corresponding to a given type of modification for the `IlvChartDataSetListener` class can be found in *Listeners on Data Sets* on page 139 and for the `IlvChartDataListener` class in *Listeners on Chart Data Objects* on page 139.
3. Add an instance of the created subclass to the data set (or to the chart data object).

Example of a User-Defined Listener

We are now going to show you how to define a listener that performs specific tasks when new data items are added to a data set. We will use the Temperatures Chart again (see *Creating a Simple Cartesian Chart* on page 103).

In this example, we want the chart to indicate when temperatures for each data set fall outside of a given interval. We are going to add two cursors to the Temperatures Chart that indicate the minimum and maximum values of the interval. When a new temperature that is added to a data set is outside of this interval, a message will be printed on the standard output. To do this, we can define a specific listener (inherited from `IlvChartDataSetListener`) to be set on each temperatures data set.

The complete source code of this example can be found in the `listener.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory. The example shows only the part of the code that is specific to creating the dedicated listener.

Defining the TemperatureDataSetListener Class

The following code is used to declare the TemperatureDataSetListener class:

```
class TemperatureDataSetListener
    : public IlvChartDataSetListener
{
public:
    TemperatureDataSetListener(IlDouble min,
                               IlDouble max,
                               IlvChartGraphic* chart = 0,
                               IlvPalette* cursorPalette = 0);

    virtual void dataPointAdded(const IlvChartDataSet* dataSet,
                                IlUInt position);

protected:
    IlDouble      _min;
    IlDouble      _max;
    IlvChartGraphic* _chart;
};
```

Only the dataPointAdded method is subclassed since we want to perform a specific task when a new data item is added, and nothing special when other types of modifications are made to the data set.

The constructor is implemented as follows:

```
TemperatureDataSetListener::
TemperatureDataSetListener(IlDouble min,
                           IlDouble max,
                           IlvChartGraphic* chart,
                           IlvPalette* cursorPalette)
    : IlvChartDataSetListener(),
      _min(min),
      _max(max)
{
    if (chart) {
        IlvDisplay* dpy = chart->getDisplay();
        IlvPalette* palette = cursorPalette ? cursorPalette
        : dpy->getPalette(dpy->getColor("white"), 0);
        chart->addOrdinateCursor(_min, palette);
        chart->addOrdinateCursor(_max, palette);
    }
}
```

The constructor takes the minimum and maximum values of the considered interval as parameters. When a temperature that is outside of this interval is added, a message is printed. If a chart is specified as a parameter, cursors will be drawn in the chart at the minimum and maximum values of the interval.

The implementation of the `dataPointAdded` method is the following:

```
void
TemperatureDataSetListener::dataPointAdded(const IlvChartDataSet* dataSet,
                                           IUInt position)
{
    IlvDoublePoint dataPoint;
    dataSet->getPoint(position, dataPoint);
    if (dataPoint.y() < _min)
        cout << dataSet->getName() << ": Temperature at the index "
              << position << " = " << dataPoint.y() << " degrees ( < "
              << _min << " )" << endl;
    if (dataPoint.y() > _max)
        cout << dataSet->getName() << ": Temperature at the index "
              << position << " = " << dataPoint.y() << " degrees ( > "
              << _max << " )" << endl;
}
```

The method compares the new temperature value that is added to the data set with the minimum and the maximum values defined. A specific message is printed on the standard output when the temperature value is smaller than the minimum and when the temperature value is greater than the maximum.

Adding a TemperatureDataSetListener Object to the Temperatures Data Sets

The listener to be added to the temperatures data sets is created by the following code:

```
TemperatureDataSetListener* listener =
    new TemperatureDataSetListener(10, 22, chart);
```

The created listener is set on all the temperatures data sets that are defined. It is set on `dataSets[0]` that corresponds to the morning mean temperatures data set and on `dataSets[1]` that corresponds to the afternoon mean temperatures data set.

```
dataSets[0]->addListener(listener);
dataSets[1]->addListener(listener);
```

Note: The listener must be set on the temperatures data sets before putting the data into the data sets. Otherwise, the message indicating whether a temperature is outside of the defined interval will not be printed when the temperature is added to a data set.

Now that the listener has been added, the Temperatures Chart appears as shown in Figure 6.4:

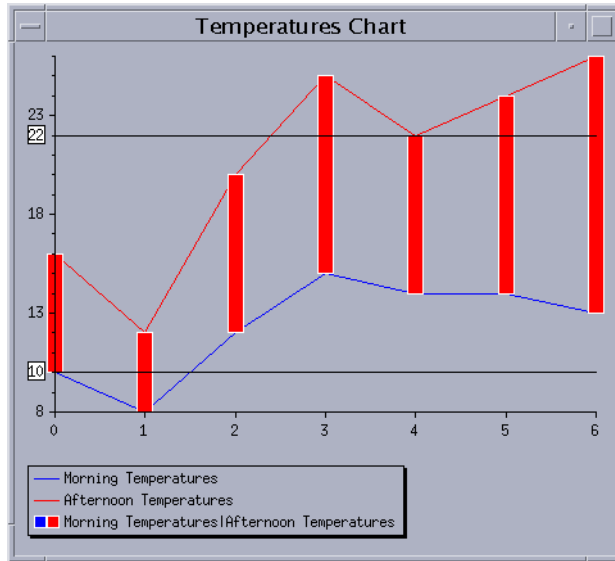


Figure 6.4 The Temperatures Chart with a Listener Set On the Temperatures Data Sets

Notice that a cursor has been added at the minimum value (10) and at the maximum value (22) which are passed as parameters to the constructor of the listener.

The messages that are printed to the standard output are the following:

```
Morning Temperatures: Temperature at the index 1 = 8 degrees ( < 10 )
Afternoon Temperatures: Temperature at the index 3 = 25 degrees ( > 22 )
Afternoon Temperatures: Temperature at the index 5 = 24 degrees ( > 22 )
Afternoon Temperatures: Temperature at the index 6 = 26 degrees ( > 22 )
```

Chart Layout

This chapter provides detailed information on the chart layout. You can find information on the following topics:

- ◆ *Computing the Chart Layout*
- ◆ *Setting General Properties of a Chart Layout Object*
- ◆ *Getting and Setting the Chart Layout Object of a Chart*

Computing the Chart Layout

The global layout of a chart is computed by a dedicated object called the *layout* object. When we speak about the global layout of the chart, we are referring to the position of the different areas of the chart within its bounding box. We distinguish three areas within the bounding box of a chart:

- ◆ The *drawing area* is the area where the drawing is performed. All the graphical elements that make up a chart (that is, the graphical representations of data, scales, grids, and cursors) are drawn within this area. The drawing area is defined by margins relative to the bounding box of the chart.
- ◆ The *data display area* is the area where the data are displayed. No data points can be displayed outside of this area. The data display area lies inside the drawing area.

- ◆ The *graph area* represents the extent of all the graphical elements that make up a chart (that is, all the graphical representations of data, scales, grids, and cursors of the chart). This area lies inside the drawing area and contains the data display area.

By default, the graph area and the data display area are automatically computed from the drawing area so that the graph area takes up the maximum amount of available space within the drawing area. For Cartesian charts, the graph area will be equal to the drawing area. For polar charts, the data display area must be a square. In this case, the data display area will be the largest square possible so that the corresponding graph area lies within the drawing area.

Figure 7.1 shows an example of these areas for a Cartesian chart. The bounding box of the chart is the largest rectangle. The drawing area, graph area, and data display area are positioned within the bounding box. The drawing area and the graph area are equivalent so they appear as the same rectangle (the second largest rectangle in the figure). You can see that the graphical representations of data and the scales are all contained within the graph area. The smallest rectangle shown in the figure is the data display area. This area contains the ranges of data represented by all the scales of the chart.

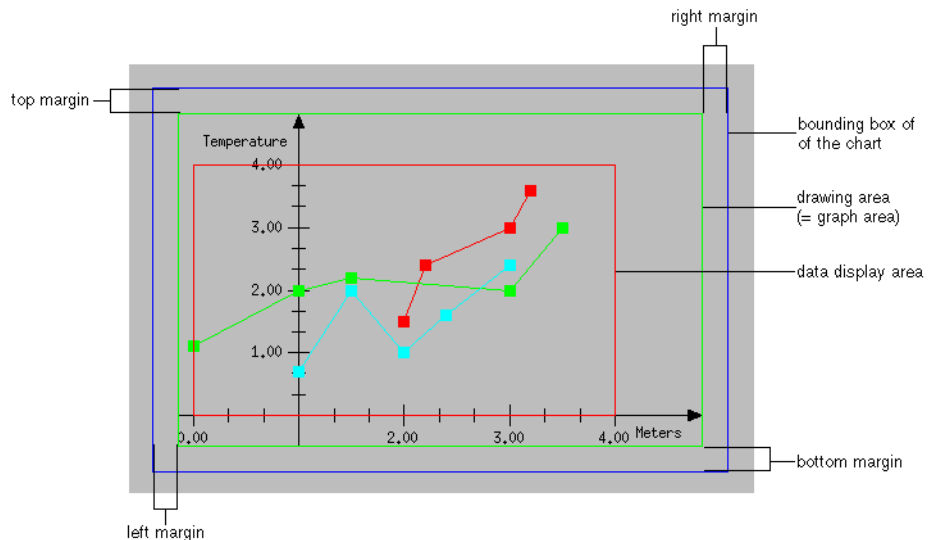


Figure 7.1 Areas within the Bounding Box of a Cartesian Chart

Figure 7.2 shows an example of these areas for a polar chart. You can notice that the drawing area is not equivalent to the graph area and that the data display area is a square.

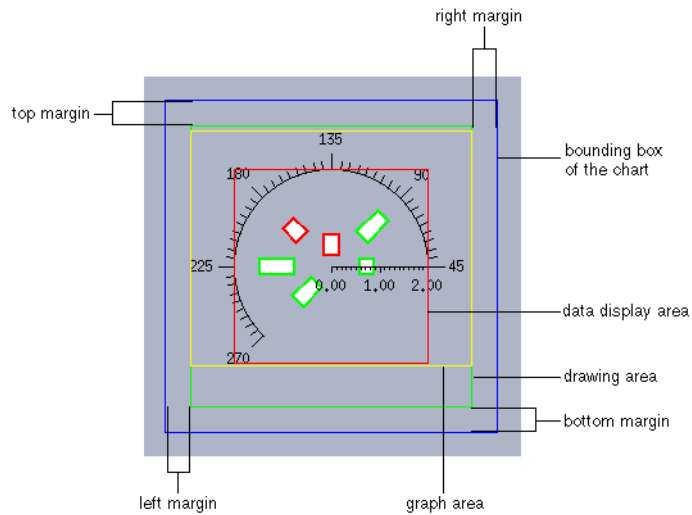


Figure 7.2 Areas within the Bounding Box of a Polar Char

The base class used to represent a layout object is the `IlvChartLayout` class. No subclass is defined in the Charts Library.

Setting General Properties of a Chart Layout Object

The following properties are defined for a layout object:

Property	Methods	Default Value
Areas Computation Properties		
Graph and Data Display Areas Automatically Computed	<code>isAutoLayout</code> <code>setAutoLayout</code>	<code>IlvTrue</code>
Graph Area Automatically Computed	<code>isAutoGraphArea</code> <code>setAutoGraphArea</code>	<code>IlvTrue</code>
Data Display Area Automatically Computed	<code>isAutoDataDisplayArea</code> <code>setAutoDataDisplayArea</code>	<code>IlvTrue</code>
Areas Positioning Properties		

Property	Methods	Default Value
Drawing Area Defined by: Margins Between the Chart Bounding Box and the Drawing Area	<code>getLeftMargin</code> <code>setLeftMargin</code> <code>getRightMargin</code> <code>setRightMargin</code> <code>getTopMargin</code> <code>setTopMargin</code> <code>getBottomMargin</code> <code>setBottomMargin</code>	 0 0 0 0
Graph Area Defined by Hand Using: Margins Between the Drawing Area and the Graph Area A Rectangle Corresponding to the Graph Area	<code>getGraphAreaRelatively</code> <code>setGraphAreaRelatively</code> <code>getGraphArea</code> <code>setGraphArea</code>	
Data Display Area Defined by Hand Using: Margins Between the Drawing Area and the Data Display Area A Rectangle Corresponding to the Data Display Area	<code>getDataDisplayAreaRelatively</code> <code>setDataDisplayAreaRelatively</code> <code>getDataDisplayArea</code> <code>setDataDisplayArea</code>	

By default, the graph area and the data display area are automatically computed from the drawing area. However, the graph area or the data display area can be fixed by hand. If the graph area is fixed by hand, the data display area will be computed automatically from the fixed graph area. Similarly, if the data display area is fixed by hand, the graph area will be automatically computed from the fixed data display area.

Note: Be careful when you fix the data display area by hand. If the specified data display area is not small enough, the computed graph area can be larger than the drawing area and thus a part of the drawing of the chart may be not visible on the screen.

You can fix the graph area or the data display area by hand in two ways:

- ◆ By specifying margins relative to the drawing area

This can be done by means of the `IlvChartLayout::setGraphAreaRelatively` or `IlvChartLayout::setDataDisplayAreaRelatively` methods. In this case, the area that is fixed will be updated if the drawing area is modified (for example when the margins between the drawing area and the chart bounding box are changed).

- ◆ By setting the rectangle corresponding to the area to be fixed directly

This can be done by means of the `IlvChartLayout::setGraphArea` or `IlvChartLayout::setDataDisplayArea` methods. In this case, the area that is fixed does not depend on the drawing area and is not updated when the drawing area is modified.

Getting and Setting the Chart Layout Object of a Chart

When you create a chart object, an `IlvChartLayout` object is created by default to handle the computation of the layout of the chart.

This layout object can be accessed by means of the `getLayout` method. You can change the layout object that is used by default by means of the `setLayout` method.

Data Display

This chapter provides detailed information on data display for the charts. You can find information on the following topics:

- ◆ *Drawing the Graphical Representations of Data*
- ◆ *Using Single Displayers*
- ◆ *Using Composite Displayers*
- ◆ *Adding a Displayer to a Chart*
- ◆ *Customizing Data Display*

Drawing the Graphical Representations of Data

Each graphical representation of data is displayed within a chart by a dedicated object called a *displayer*.

The base class used to represent a displayer is the `IlvAbstractChartDisplayer` class.

The displayers are divided into two categories:

- ◆ **Single displayers** display data with a single, basic rendering shape. Examples of such displayers are the displayers representing data with markers, polylines, bars, and so on.

- ◆ **Composite displayers** display data using a combination of several rendering shapes. These displayers can display more complex graphical representations of data. Examples of such displayers are the displayers representing data with a marked polyline, stacked bars, and so on.

Composite displayers are defined as a combination of other displayers, which can be either single displayers or other composite displayers. For example, a marked polyline displayer is composed of two single displayers (one representing data with a polyline and one representing data with markers). A stacked bar displayer is composed of several single displayers representing data with bars.

Figure 8.1 shows the hierarchy of all the displayers defined in the Charts Library. On the left are the single displayers and on the right are the composite displayers.

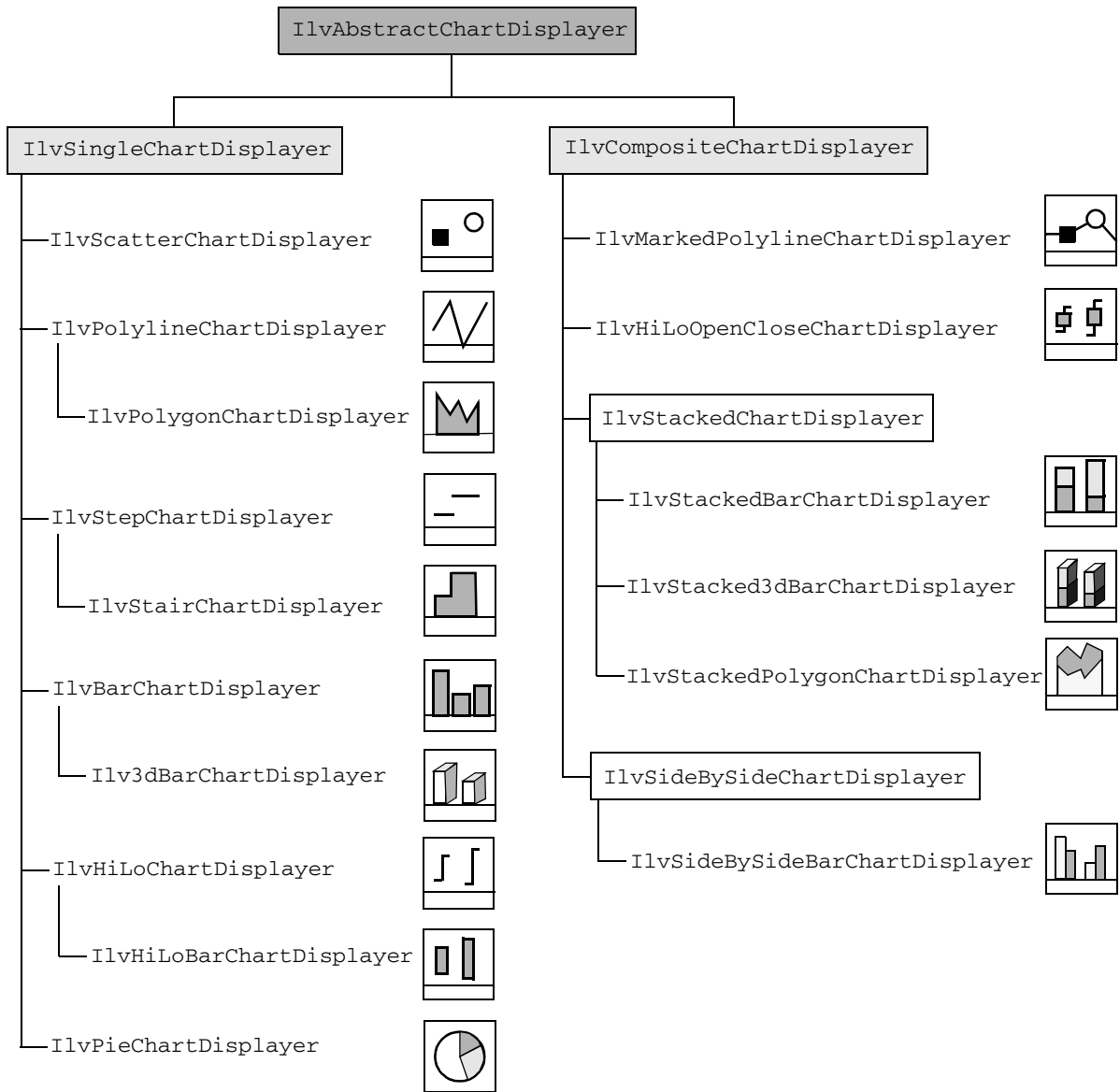


Figure 8.1 Hierarchy of Displayers in the Charts Library

A displayer can be used to display one or more data sets, depending on the type of the displayer. For example, a displayer that represents data with bars displays a unique data set,

and a displayer that represents data with a high-low representation displays two data sets. For a given displayer, it is important to distinguish between two kinds of data sets:

- ◆ **Real data sets** are the data sets that are to be displayed by the displayer and that you set on the displayer. These data sets should be stored in the chart data object that is set on the chart object using the displayer.
- ◆ **Virtual data sets** are the data sets that are constructed just when needed for drawing the graphical representation of the real data sets that you want to display with the displayer. Virtual data sets are constructed only in displayers that cannot directly display the real data sets and need to put the data into another format before displaying them. For example, a pie displayer needs a specific virtual data set to internally transform the real data set into a set of angle values that can be then displayed.

Note: If virtual data sets are defined for a given displayer, it is these data sets, constructed from the real data sets that are set on the displayer, that will actually be displayed by the displayer. Otherwise, the displayer will directly display the real data sets that are set on the displayer.

Setting General Properties

The following properties are defined for all the displayers:

Property	Methods	Default Value
Drawing Properties		
Visibility	isVisible setVisible	IlvTrue
Palette	getPalette setPalette	0
Palette Foreground	getForeground setForeground	0
Palette Background	getBackground setBackground	0
Miscellaneous		
Name	getName setName	0
Flags	getFlags setFlags	0

Property	Methods	Default Value
Legend Text	getLegendText setLegendText	0
Parent Displayer	getParentDisplayer setParentDisplayer	0

If the considered displayer is a single displayer, a unique palette will be set by means of the `IlvAbstractChartDisplayer::setPalette` method. Otherwise, if the displayer is a composite displayer, a palette can be set for each child displayer that makes up the composite displayer.

If no palette has been specifically defined for a given displayer, the displayer will use the palette set on the chart object that uses the displayer.

The Legend Text property is used to define the text of a legend for the displayer. If no legend text is defined, the name of the real data set(s) displayed by the displayer will be used.

If a displayer is one of the displayers that make up a composite displayer, the method `IlvAbstractChartDisplayer::getParentDisplayer` returns the composite displayer.

Using Single Displayers

The base class used to represent a single displayer is the `IlvSingleChartDisplayer` class.

Setting General Properties

The following property is defined for all the single displayers:

Property	Methods	Default Value
Drawn Filled	isDrawingFilled drawFilled	IlvTrue

If the Drawn Filled property is set to `IlvTrue`, the shape that is rendered is filled with the background color of the palette. Otherwise, the shape is simply outlined with the foreground color of the palette.

Predefined Single Displayers


The following sections describe the single displayers of the Charts Library. For each displayer, you will find a table that describes the conditions for using the displayer, including the number of real data sets that are displayed with the displayer and whether the displayer can be used no matter what the type of the projection is.

You can find information on the following single displayers:

- ◆ *Scatter Displayer*
- ◆ *Polyline Displayer*
- ◆ *Polygon Displayer*
- ◆ *Step Displayer*
- ◆ *Stair Displayer*
- ◆ *Bar Displayer*
- ◆ *3D Bar Displayer*
- ◆ *High-Low Displayer*
- ◆ *High-Low Bar Displayer*
- ◆ *Pie Displayer*

Scatter Displayer

A scatter displayer has the following basic characteristics:

Class	IlvScatterChartDisplayer
Category	Single
Number of real data sets displayed	1
Can be used with all types of projections	Yes
Items drawn	Markers 

The following properties are specific to a scatter displayer:

Property	Methods	Default Value
Marker Type	getMarker setMarker	IlvMarkerFilledSquare
Marker Size	getMarkerSize setMarkerSize	IlvDefaultMarkerSize

Figure 8.2 illustrates the fact that a scatter displayer can be used with all types of projections. Data sets are represented by scatter displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

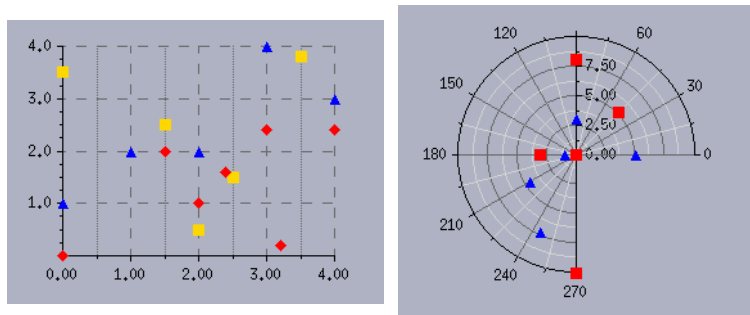


Figure 8.2 Scatter Displays in a Cartesian Chart and in a Polar Chart


We can use the scatter displayer displaying the blue triangles as an example. We can create this displayer by using the following code:

```
IlvScatterChartDisplayer* displayer =
    new IlvScatterChartDisplayer (IlvMarkerFilledTriangle,
                                IlvDefaultMarkerSize);
displayer->setForeground(dpy->getColor("blue"));
```

Note: The *Drawn Filled* property has no meaning for a scatter displayer. The markers are simply drawn with the foreground color of the defined palette.

Polyline Displayer

A polyline displayer has the following basic characteristics:

Class	IlvPolylineChartDisplayer
Category	Single
Number of real data sets displayed	1
Can be used with all types of projections	Yes
Items drawn	Polyline 

The following property is specific to a polyline displayer:

Property	Methods	Default Value
Projected Points Palette	getProjectedPointsPalette setProjectedPointsPalette	0

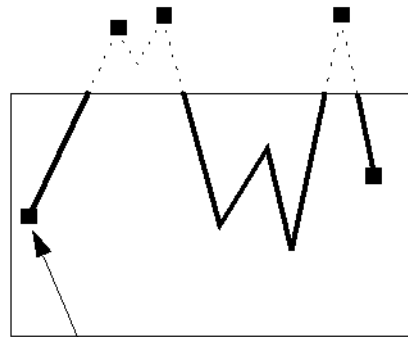
By default, any data points that should appear outside of the data display area (if such data points exist) are not drawn. The parts of the polyline extending outside of the data display area are simply clipped by the data display area. The drawing on the left in Figure 8.3 shows how a polyline can be clipped. The screen points corresponding to the data points to be displayed are represented by black squares. Only the parts of the polyline displayed in bold will appear on the screen.

The data points that should appear outside of the data display area can be projected onto the limits of the data display area if desired. Data points that are outside of the data display area (out-of-bounds data points) are projected only if a horizontal projection and/or a vertical projection of out-of-bounds data points are requested at the level of the chart object that uses the displayer. A horizontal projection can be requested on a chart by the method `IlvChartGraphic::setProjectHorizontally`. A vertical projection can be requested on a chart by the method `IlvChartGraphic::setProjectVertically`.

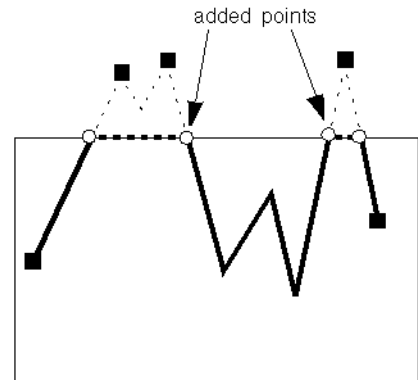
The drawing on the right in Figure 8.3 shows how out-of-bounds data points are handled. The parts of the polyline extending outside of the data display area are projected. Points are added (those shown by white circles) in order to close the polyline that was truncated in the drawing on the left. The polyline then appears to be continuous with the projected parts following the limits of the data display area.

These projected parts are displayed with the palette returned by the `IlvPolylineChartDisplayer::getProjectedPointsPalette` method. This method returns a pointer to the palette object that has been set to display out-of-bounds data points for the current polyline displayer, if such a palette has been set. Otherwise, it returns a pointer to the palette object that has been set to display out-of-bounds data points in the chart object that uses the current polyline displayer.

Without projecting out-of-bounds data points



With projecting out-of-bounds data points



screen point corresponding to a data point to be displayed

Figure 8.3 Out-of-Bounds Data Point Projection

Figure 8.4 illustrates the fact that a polyline displayer can be used with all types of projections. Data sets are represented by polyline displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

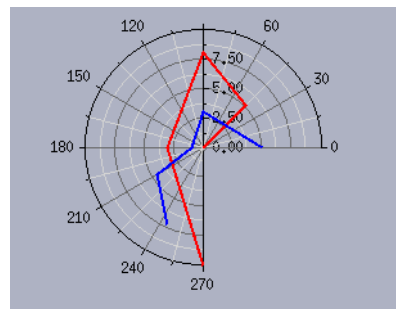
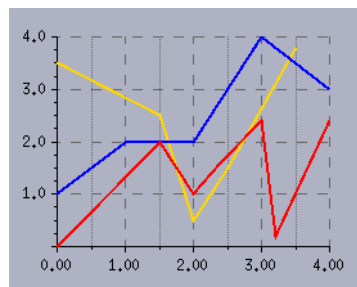


Figure 8.4 Polyline Displayers in a Cartesian Chart and in a Polar Chart

We can use the polyline displayer displaying the blue polyline as an example. We can create this displayer by using the following code:

```
IlvPolylineChartDisplayer* displayer = new IlvPolylineChartDisplayer();
displayer->setForeground(dpy->getColor("blue"));
```

Note: The *Drawn Filled* property has no meaning for the polyline displayer. The polyline is simply drawn with the foreground color of the defined palette.

Polygon Displayer

A polygon displayer has the following basic characteristics:


Class	IlvPolygonChartDisplayer
Category	Single
Inherits from	IlvPolylineChartDisplayer
Number of real data sets displayed	1
Can be used with all types of projections	Yes
Items drawn	Polygon 

Figure 8.5 illustrates the fact that a polygon displayer can be used with all types of projections. Data sets are represented by polygon displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

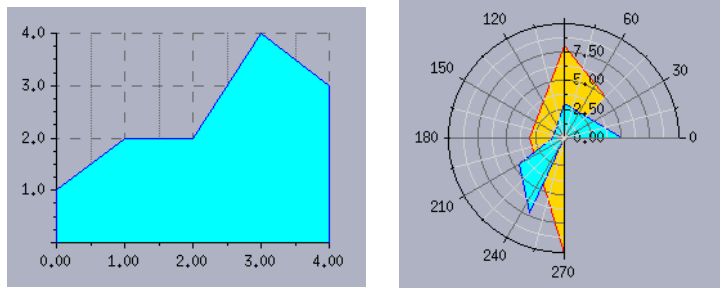


Figure 8.5 Polygon Displayers in a Cartesian Chart and in a Polar Chart

We can use the polygon displayer displaying the cyan polygon as an example. We can create this displayer by using the following code:

```
IlvPolygonChartDisplayer* displayer = new IlvPolygonChartDisplayer();
displayer->setForeground(dpy->getColor("blue"));
displayer->setBackground(dpy->getColor("cyan"));
```

Note: By default, the polygon is outlined with the foreground color and filled with the background color of the defined palette. It will be displayed only with the outline if the *Drawn Filled* property is set to `IlvFalse`.

Step Displayer

A step displayer has the following basic characteristics:


Class	<code>IlvStepChartDisplayer</code>
Category	Single
Number of real data sets displayed	1
Can be used with all types of projections	Yes
Items drawn	Steps 

Figure 8.6 illustrates the fact that a step displayer can be used with all types of projections. Data sets are represented by step displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

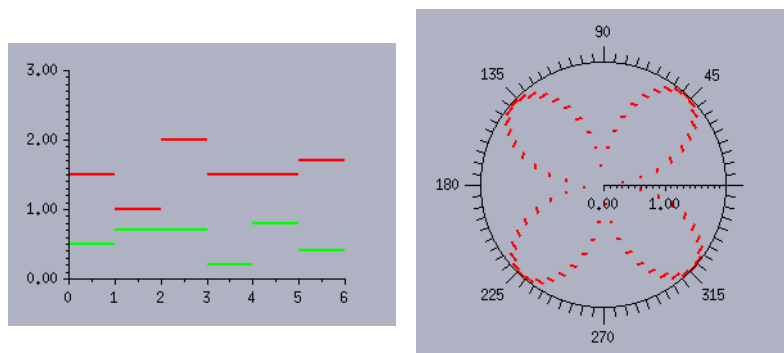


Figure 8.6 Step Displayers in a Cartesian Chart and in a Polar Chart

We can use the step displayer displaying the red steps as an example. We can create this displayer by using the following code:

```
IlvStepChartDisplayer* displayer = new IlvStepChartDisplayer();
displayer->setForeground(dpy->getColor("red"));
```

Note: The *Drawn Filled* property has no meaning for the step displayer. The steps are simply drawn with the foreground color of the defined palette.

Stair Displayer

A stair displayer has the following basic characteristics:


Class	IlvStairChartDisplayer
Category	Single
Inherits from	IlvStepChartDisplayer
Number of real data sets displayed	1
Can be used with all types of projections	Yes
Items drawn	Stairs 

Figure 8.7 illustrates the fact that a stair displayer can be used with all types of projections. Data sets are represented by stair displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

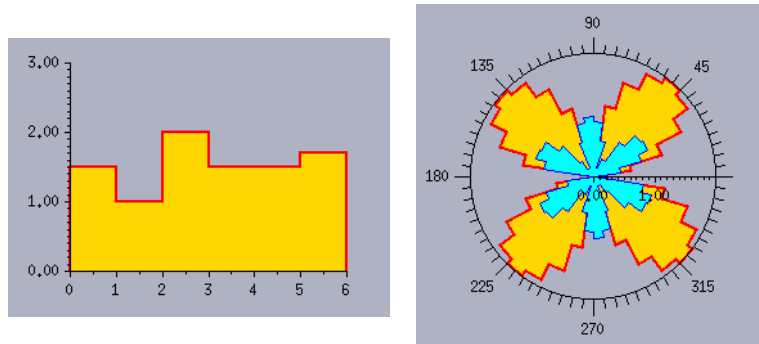


Figure 8.7 Stair Displayers in a Cartesian Chart and in a Polar Chart

We can use the stair displayer displaying the gold stairs as an example. We can create this displayer by using the following code:

```
IlvPalette* palette = dpy->getPalette(dpy->getColor("gold"),  
                                     dpy->getColor("red"),  
                                     0,0,0,0,2);  
IlvStairChartDisplayer* displayer = new IlvStairChartDisplayer(palette);
```

Note: By default, the stairs are outlined with the foreground color and filled with the background color of the defined palette. They will be displayed only with the outline if the *Drawn Filled* property is set to `IlvFalse`.

Bar Displayer

A bar displayer has the following basic characteristics:

Class	<code>IlvBarChartDisplayer</code>
Category	Single
Number of real data sets displayed	1
Can be used with all types of projections	Yes
Items drawn	Bars 

The following property is specific to a bar displayer:

Property	Methods	Default Value
Width of a Bar	<code>getWidth</code> <code>setWidth</code>	<code>IlvChartDisplayerWidth</code>
Width Percentage of a Bar	<code>getWidthPercent</code> <code>setWidthPercent</code>	100

If a width percentage is set, the width of the bars is no longer constant but proportional to the zoom level of the chart.

Figure 8.8 illustrates the fact that a bar displayer can be used with all types of projections. Data sets are represented by bar displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

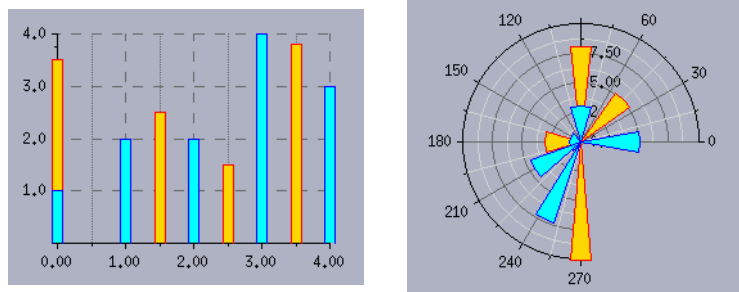


Figure 8.8 Bar Displayers in a Cartesian Chart and in a Polar Chart


We can use the bar displayer displaying the gold bars as an example. We can create this displayer by using the following code:

```
IlvBarChartDisplayer* displayer =
    new IlvBarChartDisplayer(IlvChartDisplayerWidth);
displayer->setForeground(dpy->getColor("red"));
displayer->setBackground(dpy->getColor("gold"));
```

Note: By default, the bars are outlined with the foreground color and filled with the background color the defined palette. They will be displayed only with the outline if the *Drawn Filled* property is set to `IlvFalse`.

3D Bar Displayer

A 3D bar displayer has the following basic characteristics:

Class	Ilv3dBarChartDisplayer
Category	Single
Inherits from	IlvBarChartDisplayer
Number of real data sets displayed	1
Can be used with all types of projections	Yes
Items drawn	3D bars 

The following property is specific to a 3D bar displayer:

Property	Methods	Default Value
Depth of a 3D Bar	getDepth setDepth	IlvChartDisplayerDepth

Figure 8.9 illustrates the fact that a 3D bar displayer can be used with all types of projections. Data sets are represented by 3D bar displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

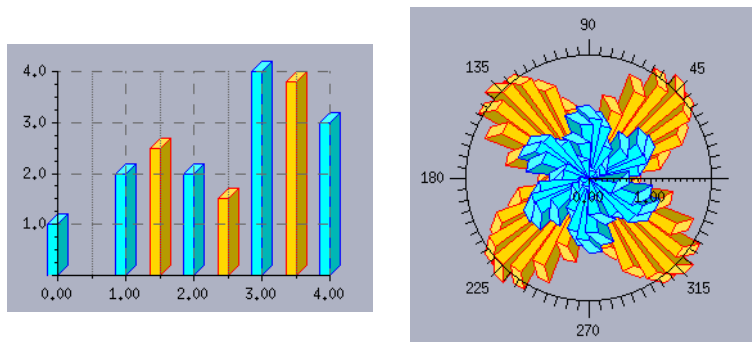


Figure 8.9 3D Bar Displayers in a Cartesian Chart and in a Polar Chart


We can use the 3D bar displayer displaying the gold 3D bars as an example. We can create this displayer by using the following code:

```
IlvPalette* palette = dpy->getPalette(dpy->getColor("gold"),
                                     dpy->getColor("red"));
Ilv3dBarChartDisplayer* displayer =
    new Ilv3dBarChartDisplayer(IlvChartDisplayerWidth,
                              IlvChartDisplayerDepth,
                              palette);
```

Note: By default, the 3D bars are outlined with the foreground color, the front face is filled with the background color, and the top and the side faces are filled with shadow colors computed from the background color. The 3D bars will be displayed only with the outline if the *Drawn Filled* property is set to `IlvFalse`.

High-Low Displayer

A high-low displayer has the following basic characteristics:

Class	<code>IlvHiLoChartDisplayer</code>
Category	Single
Number of real data sets displayed	2
Can be used with all types of projections	Yes
Items drawn	High-low items 

The high-low displayer displays two data sets. The first data set is composed of the low values and the second data set is composed of the high values. A high-low item is drawn between each pair of low-high values, the low values being taken in order from the first data set and the high values being taken in order from the second data set.

The following properties are specific to a high-low displayer:

Property	Methods	Default Value
Width of a High-low Item	getWidth setWidth	IlvChartDisplayerWidth
Width percentage of a High-low Item	getWidthPercent setWidthPercent	100
Rise Palette	getRisePalette setRisePalette	0
Fall Palette	getFallPalette setFallPalette	0

If a width percentage is set, the width of the items is no longer constant but proportional to the zoom level of the chart.

Two palettes are defined: a rise palette and a fall palette. The rise palette is used to draw the high-low items for which the corresponding low value is less than the high value. The fall palette is used to draw the high-low items for which the corresponding low value is greater than the high value.

Figure 8.10 illustrates the fact that a high-low displayer can be used with all types of projections. Data sets are represented by a high-low displayer in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

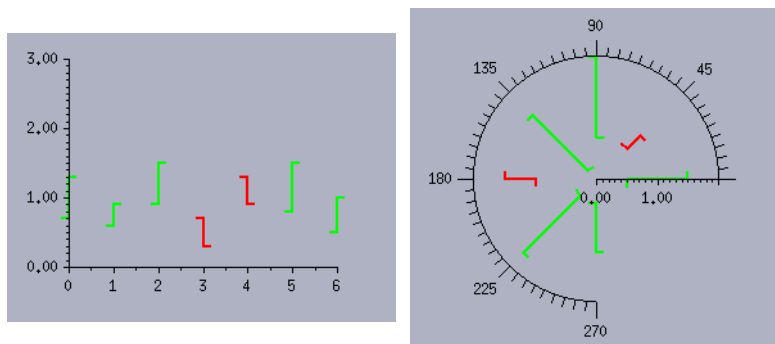


Figure 8.10 High-Low Displayers in a Cartesian Chart and in a Polar Chart


We can use the high-low displayer displaying the green and red high-low items as an example. We can create this displayer by using the following code:

```
IlvPalette* risePal = dpy->getPalette(dpy->getColor("white"),
                                     dpy->getColor("green"),0,0,0,0,2);
IlvPalette* fallPal = dpy->getPalette(dpy->getColor("white"),
                                     dpy->getColor("red"),0,0,0,0,2);
IlvHiLoChartDisplayer* displayer =
    new IlvHiLoChartDisplayer(IlvChartDisplayerWidth,
                              risePal,
                              fallPal);
```

Note: *The Drawn Filled property has no meaning for the high-low displayer. The high-low items are simply drawn with the foreground color of the rise and fall palettes defined.*

High-Low Bar Displayer

A high-low bar displayer has the following basic characteristics:

Class	IlvHiLoBarChartDisplayer
Category	Single
Inherits from	IlvHiLoChartDisplayer
Number of real data sets displayed	2
Can be used with all types of projections	Yes
Items drawn	High-low bar items 

As with the high-low displayer, the high-low bar displayer displays two data sets. The first data set is composed of the low values and the second data set is composed of the high values. A high-low bar item is drawn between each pair of low-high values, the low values being taken in order from the first data set and the high values being taken in order from the second data set.

Figure 8.11 illustrates the fact that a high-low bar displayer can be used with all types of projections. Data sets are represented by a high-low bar displayer in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

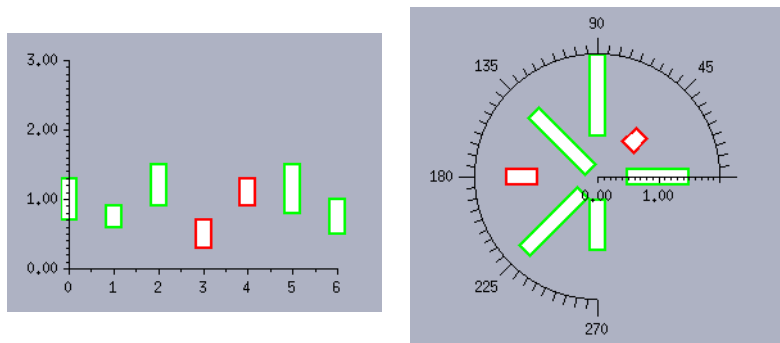


Figure 8.11 High-Low Bar Displays in a Cartesian Chart and in a Polar Chart


We can use the high-low bar displayer displaying the white high-low bar items outlined in green and red as an example. We can create this displayer by using the following code:

```
IlvPalette* risePal = dpy->getPalette(dpy->getColor("white"),
                                     dpy->getColor("green"),0,0,0,0,2);
IlvPalette* fallPal = dpy->getPalette(dpy->getColor("white"),
                                     dpy->getColor("red"),0,0,0,0,2);
IlvHiLoBarChartDisplayer* displayer =
    new IlvHiLoBarChartDisplayer(IlvChartDisplayerWidth,
                                risePal,
                                fallPal);
```

Note: By default, the high-low bar items are outlined with the foreground color of the defined rise and fall palettes and are filled with the background color of these palettes. The high-low bar items will be displayed only with the outline if the `Drawn Filled` property is set to `IlvFalse`.

Pie Displayer

A pie displayer has the following basic characteristics:

Class	<code>IlvPieChartDisplayer</code>
Category	Single
Number of real data sets visualized	1
Can be used with all types of projections	No (Use only with a polar projection)
Items drawn	Slices 

The pie displayer displays a unique data set. Only the y-values of the data points represented by the data set are considered. It is these values that will be represented as slices of the pie. For this reason, it is better to use a data set defined as a set of values (an instance of the `IlvChartYValueSet` class) to store the data you want to display with a pie. However, it is also possible to use another type of data set (a set-of-points data set or a function data set). Just remember that the values that will be represented as slices of the pie are the y-values of the data points represented by your data set.

The y-values of the data points cannot be displayed directly by the pie displayer, since angle values are needed to draw these values as slices. Internally, the pie displayer uses a specific virtual data set that is composed of data points expressed in polar coordinates (θ, ρ) computed from the initial data points of the real data set. The computed data points lie on the extremities of the arcs of the slices. It is this virtual data set that will actually be displayed by the pie displayer.

Since the data points that will be displayed by the pie displayer are expressed in polar coordinates, the pie displayer can only be used with a polar projection. This means that a pie displayer should only be added to a chart object using a polar projector. A dedicated subclass of `IlvPolarChart` has been implemented to facilitate the use of pie displayers. The `IlvPieChartGraphic` class is used to create a chart object directly instantiated to display pies. This class encapsulates the creation of the pie displayers.

The following properties are specific to a pie displayer:

Property	Methods	Default Value
General Properties		
Radius of the Pie	<code>getRadius</code> <code>setRadius</code>	0
Angle at Which the First Slice is Drawn	<code>getStartingAngle</code> <code>setStartingAngle</code>	0
Angle Range of the Pie	<code>getRange</code> <code>setRange</code>	360
Offset Applied to Torn Off Slices	<code>getTearOffDelta</code> <code>setTearOffDelta</code>	20
Offset Between the Slices and the Graphic Objects Added to the Slices	<code>getOffset</code> <code>setOffset</code>	(0, 0)
Slice Properties		
Palette of a Slice	<code>getSlicePalette</code> <code>setSlicePalette</code>	0

Property	Methods	Default Value
Slice Torn Off	isSliceTornOff tearOffSlice	IlvFalse
Graphic Object Drawn in Addition to a Slice	getSliceGraphic setSliceGraphic	0
Legend Text for a Slice	getSliceLegendText setSliceLegendText	0

The properties related to a given slice are stored in a dedicated object called a *slice information* object. The slice information objects added to the slices are instances of the `IlvPieSliceInfo` class by default. A graphic object can be added to a given slice to display information related to the slice (for example, an annotation).

Example

Figure 8.12 shows a data set represented by a pie displayer in a pie chart object. A slice information object storing a label (an instance of the `IlvLabel` class) has been added to each slice of the pie.

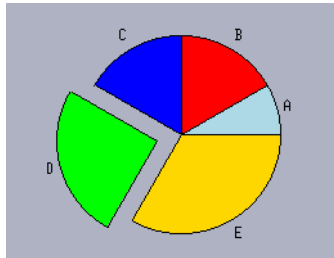


Figure 8.12 Pie Displayer in a Pie Chart (Using a Polar Projection)

This section contains a description of all the steps required to display this pie chart. We use an `IlvPieChartGraphic` object to create the chart object that displays the pie chart. Since the `IlvPieChartGraphic` object is already instantiated to display pie charts, it is easier to use this class instead of an `IlvChartGraphic` object. However, if you want to, you can create a pie displayer by hand and add it to the chart object you want, provided the chart object uses a polar projection.

The complete source code of this example can be found in the `pie.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory.

Creating the Data Set

We can create the data set for this chart by using the following code:

```
IlvChartYValueSet* dataSet = new IlvChartYValueSet();
dataSet->addValue(1.);
dataSet->addValue(2.);
dataSet->addValue(2.);
dataSet->addValue(3.);
dataSet->addValue(4.);
```

Creating the Pie Chart

We can create the chart object by using the following code:

```
IlvPieChartGraphic* chart = new IlvPieChartGraphic(display,
                                                    IlvRect(10, 10, 450, 300));
```

The `IlvRect` object passed as a parameter corresponds to the bounding box of the created chart.

Adding the Data Set to the Chart Data Object

The data set is added to the chart data object set on the chart object by using the following code:

```
chart->getData()->addDataSet(dataSet);
```

Creating and Adding the Pie Displayer

The pie displayer is created and added to the chart object using the following code:

```
chart->addPieDisplayer(dataSet);
```

The data set that will be displayed by the pie displayer is passed as a parameter.

Customizing the Display of the Pie Slices**1. Define a different palette for each slice.**

We want to display each slice with a different color. To do so, we set a different palette for each slice by using the following code:

```
IlvPalette* bluePal = display->getPalette(0, display->getColor("lightblue"));
IlvPalette* redPal = display->getPalette(0, display->getColor("red"));
IlvPalette* darkBluePal = display->getPalette(0, display->getColor("blue"));
IlvPalette* greenPal = display->getPalette(0, display->getColor("green"));
IlvPalette* goldPal = display->getPalette(0, display->getColor("gold"));

chart->getPieDisplayer(0)->setSlicePalette(0, bluePal);
chart->getPieDisplayer(0)->setSlicePalette(1, redPal);
chart->getPieDisplayer(0)->setSlicePalette(2, darkBluePal);
chart->getPieDisplayer(0)->setSlicePalette(3, greenPal);
chart->getPieDisplayer(0)->setSlicePalette(4, goldPal);
```

Note: By default, a slice is outlined with the foreground color of the palette set on the pie displayer and is filled with the foreground color of the palette set for the slice. If no palette is set for a given slice, the slice will be filled with the background color of the palette set on the pie displayer. The slices will be displayed only with the outline if the `Drawn Filled` property is set to `IlvFalse`.

2. Define a graphic object storing a specific label for each slice:

```
chart->getPieDisplayer(0)->setSliceGraphic(0, new IlvLabel(display,0,0,"A"));
chart->getPieDisplayer(0)->setSliceGraphic(1, new IlvLabel(display,0,0,"B"));
chart->getPieDisplayer(0)->setSliceGraphic(2, new IlvLabel(display,0,0,"C"));
chart->getPieDisplayer(0)->setSliceGraphic(3, new IlvLabel(display,0,0,"D"));
chart->getPieDisplayer(0)->setSliceGraphic(4, new IlvLabel(display,0,0,"E"));
```

3. Define the offset between the slices and the added graphic objects.

```
chart->getPieDisplayer(0)->setOffset(IlvDoublePoint(0, 5));
```

The added graphic object is drawn next to a point located on the middle of the arc of the slice. The offset is defined by an angle value expressed in degrees and by a radial value expressed in pixels. The angle value is set as the abscissa of the `IlvDoublePoint` object passed as a parameter and the radial value is set as the ordinate.

4. Tear off the slice at the index 3.

```
chart->getPieDisplayer(0)->tearOffSlice(3);
```

When these steps have been completed, we obtain the chart in Figure 8.12.

Using Composite Displayers

The base class used to represent a composite displayer is the `IlvCompositeChartDisplayer` class. All composite displayers are defined as a combination of other displayers, which can be single displayers or composite displayers. The child displayers that make up a composite displayer are automatically created by the composite displayer. The type of the child displayers to be created by default is not defined at the level of the `IlvCompositeChartDisplayer` class, but is defined only in the subclasses. However, at the level of the `IlvCompositeChartDisplayer` class, you can:

- ◆ Specify a *displayer model* to define the child displayers that will be created by default.

A *displayer model* is an instance of a subclass of the `IlvAbstractChartDisplayer` class. If a displayer model is specified, the child displayers that are created by default are copies of the displayer model. A displayer model can be set on a given composite

displayer by using the `IlvCompositeChartDisplayer::setDisplayerModel` method.

Note: The displayer model is considered only if the type of the child displayers to be created by default is not specified for a given composite displayer. For example, if you set a displayer model on a stacked bar displayer (an instance of the `IlvStackedBarChartDisplayer` class), the displayer model will not be considered. It will be considered if you set it on a stacked displayer (an instance of the `IlvStackedChartDisplayer` class).

- ◆ Specify a *displayer factory* that will be used to replace the child displayers created by default by the displayers you want.

A *displayer factory* is an instance of a subclass of the `IlvChartDisplayerFactory` class. If a displayer factory is specified, the child displayers that are defined in the displayer factory will be created instead of the default child displayers. A displayer factory can be set on a given composite displayer by using the `IlvCompositeChartDisplayer::setDisplayerFactory` method.

Note: A displayer factory can be used with all types of composite displayers whether or not the type of the child displayers to be created by default is specified for the composite displayer.

Once the child displayers are created, they can be accessed by using the `IlvCompositeChartDisplayer::getDisplayer` method and can then be customized individually.

Predefined Composite Displayers


The following sections describe the composite displayers of the Charts Library. For each displayer, you will find a table that describes the conditions for using the displayer, including the number of real data sets that are displayed with the displayer and whether the displayer can be used no matter what the type of the projection is.

You can find information on the following composite displayers:

- ◆ *Marked Polyline Displayer*
- ◆ *High-Low Open-Close Displayer*
- ◆ *Stacked Displayers*
- ◆ *Side-by-Side Displayers*

Marked Polyline Displayer

A marked polyline displayer has the following basic characteristics:

Class	<code>IlvMarkedPolylineChartDisplayer</code>
Category	Composite
Number of real data sets displayed	1
Can be used with all types of projections	Yes
Number of child displayers	2 (1 scatter displayer + 1 polyline displayer)
Items drawn	Markers and/or Polyline 

The marked polyline displayer is composed of a scatter displayer (an instance of the `IlvScatterChartDisplayer` class) and a polyline displayer (an instance of the `IlvPolylineChartDisplayer` class). Each child displayer can be customized individually. You can specify the palette, the marker type, and the marker size directly for the scatter displayer. You can specify the palette directly for the polyline displayer. Use the following methods to access the child displayers:

- ◆ `IlvMarkedPolylineChartDisplayer::getMarkerDisplayer` for the scatter displayer
- ◆ `IlvMarkedPolylineChartDisplayer::getLineDisplayer` for the polyline displayer

The Charts Library also provides some methods at the level of the `IlvMarkedPolylineChartDisplayer` class that allow you to set properties defined for child displayers more quickly. The following table lists all the properties that can be set at the level of the `IlvMarkedPolylineChartDisplayer` class. (For details on all the properties that can be set on a scatter displayer, see *Scatter Displayer* on page 155. For details on all the properties that can be set on a polyline displayer, see *Polyline Displayer* on page 156.)

Property	Methods	Default Value
General Properties		
Markers Displayed	<code>isMarkerVisible</code> <code>setMarkerVisible</code>	<code>IlvTrue</code>
Polyline Displayed	<code>isLineVisible</code> <code>setLineVisible</code>	<code>IlvTrue</code>

Property	Methods	Default Value
Scatter Displayer Properties		
Palette	getMarkerPalette setMarkerPalette	0
Palette Foreground	getMarkerForeground setMarkerForeground	0
Palette Background	getMarkerBackground setMarkerBackground	0
Polyline Displayer Properties		
Palette	getLinePalette setLinePalette	0
Palette Foreground	getLineForeground setLineForeground	0
Palette Background	getLineBackground setLineBackground	0

By default, a marked polyline displayer displays a data set with both markers and a polyline. However, you can specify that only markers are to be displayed by setting the property `Polyline Displayed` to `IlvFalse`. You can specify that only a polyline is to be displayed by setting the property `Markers Displayed` to `IlvFalse`.

Figure 8.13 illustrates the fact that a marked polyline displayer can be used with all types of projections. Data sets are represented by marked polyline displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

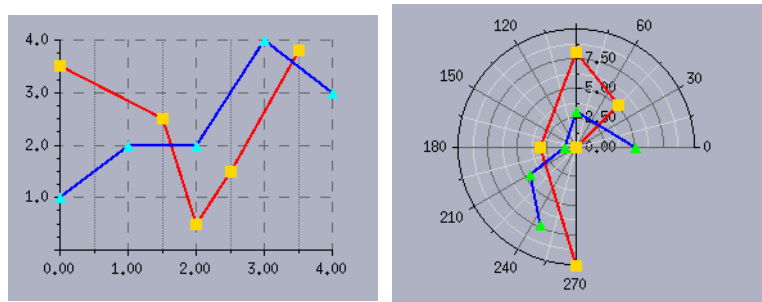



Figure 8.13 Marked Polyline Displayers in a Cartesian Chart and in a Polar Chart

We can use the marked polyline displayer displaying the red lines and gold markers as an example. To create this displayer, we can use the following code:

```
IlvMarkedPolylineChartDisplayer* displayer =
    new IlvMarkedPolylineChartDisplayer();
displayer->setMarkerForeground(dpy->getColor("gold"));
displayer->setLineForeground(dpy->getColor("red"));
```

High-Low Open-Close Displayer

A high-low open-close displayer has the following basic characteristics:

Class	IlvHiLoOpenCloseChartDisplayer
Category	Composite
Number of real data sets displayed	4
Can be used with all types of projections	Yes
Number of child displayers	2 (1 high-low displayer + 1 high-low bar displayer)
Items drawn	High-low and High-low bar items 

The high-low open-close displayer displays four data sets. The first data set is composed of the low values. The second data set is composed of the high values. The third data set is composed of the open values. The fourth data set is composed of the close values.

The first child displayer, which is a high-low displayer by default (an instance of the `IlvHiLoChartDisplayer` class), displays the first two data sets.

The second child displayer, which is a high-low bar displayer by default (an instance of the `IlvHiLoBarChartDisplayer` class), displays the next two data sets.

The default child displayers that display the two pairs of data sets can be changed by specifying a displayer factory (For details on the displayer factory, see *Using Composite Displayers* on page 171). Any type of displayer can be used provided that it inherits from the `IlvHiLoChartDisplayer` class. The following methods are used to return the two child displayers created in the high-low open-close displayer:

- ◆ `IlvHiLoOpenCloseChartDisplayer::getHiLoDisplayer` returns the displayer displaying the low-values and high-values data sets.
- ◆ `IlvHiLoOpenCloseChartDisplayer::getOpenCloseDisplayer` for the displayer displaying the open-values and close-values data sets.

Each of these two child displayers can be customized individually. For example, you can set the item width directly on each child displayer.

The Charts Library also provides some methods at the level of the `IlvHiLoOpenCloseChartDisplayer` class that allow you to set properties defined for the child displayers more quickly. The following table lists all the properties that can be set at the level of the `IlvHiLoOpenCloseChartDisplayer` class. (For details on all the properties that can be set on a high-low displayer, see *High-Low Displayer* on page 164. For details on all the properties that can be set on a high-low bar displayer, see *High-Low Bar Displayer* on page 166.)

Property	Methods	Default Value
Properties of the Displayer Displaying the Low-Values/High-Values Data Sets		
Rise Palette	<code>setHiLoRisePalette</code>	0
Fall Palette	<code>setHiLoFallPalette</code>	0
Properties of the Displayer Displaying the Open-Values/Close-Values Data Sets		
Rise Palette	<code>setOpenCloseRisePalette</code>	0
Fall Palette	<code>setOpenCloseFallPalette</code>	0

Figure 8.14 illustrates the fact that a high-low open-close displayer can be used with all types of projections. Data sets are represented by a high-low open-close displayer in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

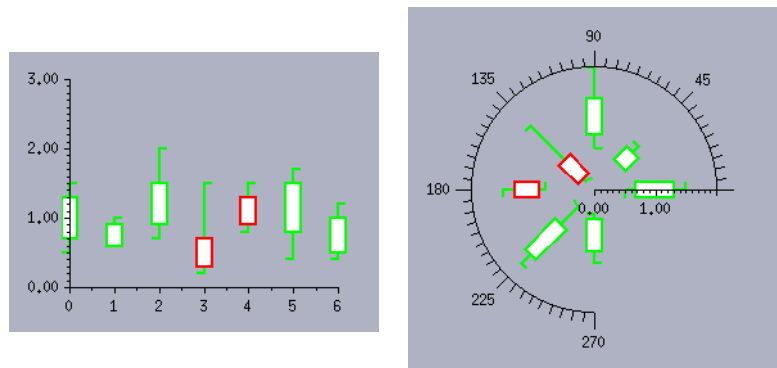


Figure 8.14 *High-Low Open-Close Displayers in a Cartesian Chart and in a Polar Chart*

We can use the high-low open-close displayer displaying the green and red high-low items and the white high-low bar items outlined in green and red as an example. To create this object, we can use the following code:

```

IlvPalette* hiLoRisePal = dpy->getPalette(dpy->getColor("white"),
                                          dpy->getColor("green"),
                                          0,0,0,0,2);
IlvPalette* openCloseRisePal = hiLoRisePal;
IlvPalette* hiLoFallPal = dpy->getPalette(dpy->getColor("white"),
                                          dpy->getColor("red"),
                                          0,0,0,0,2);
IlvPalette* openCloseFallPal = hiLoFallPal;
IlvHiLoOpenCloseChartDisplayer* displayer =
    new IlvHiLoOpenCloseChartDisplayer(IlvChartDisplayerWidth,
                                       hiLoRisePal,
                                       openCloseRisePal,
                                       hiLoFallPal,
                                       openCloseFallPal);

```

Stacked Displayers

The base class used to represent data with a stacked graphical representation is the `IlvStackedChartDisplayer` class. This type of graphical representation enables you to compare the contribution of a given data value to a total across several data sets. The type of the child displayers that are created by default is not specified at that level of the class hierarchy. However, you can define the child displayers that will be created by setting one of the following:




- ◆ A displayer model
- ◆ A displayer factory

See *Using Composite Displayers* on page 171 for details on these two objects.

The Charts Library also provides several subclasses of the `IlvStackedChartDisplayer` class that define the child displayers created by default to be stacked: bar displayers, 3D-bar displayers, and polygon displayers.

A stacked displayer has the following basic characteristics:

Class	<code>IlvStackedChartDisplayer</code>
Category	Composite
Number of real data sets displayed	As many as you want
Can be used with all types of projections	Yes

Number of child displayers	As many as the number of real data sets
Subclasses	<div style="display: flex; align-items: flex-start; gap: 20px;"> <div style="flex: 1;"> <p><code>IlvStackedBarChartDisplayer</code></p> <p><code>IlvStacked3dBarChartDisplayer</code></p> <p><code>IlvStackedPolygonChartDisplayer</code></p> </div> <div style="flex: 0.2; text-align: center;">  </div> <div style="flex: 0.2; text-align: center;">  </div> <div style="flex: 0.2; text-align: center;">  </div> </div>

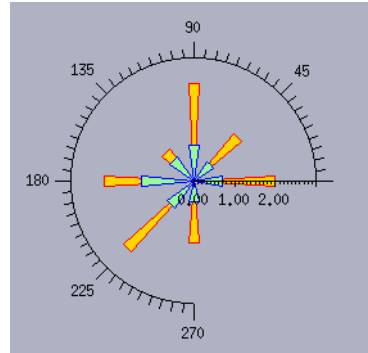
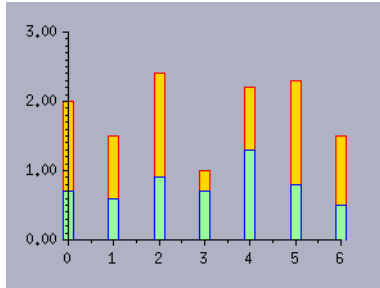
A stacked displayer displays as many data sets as you want. Within a stacked graphical representation, the first data set is represented at the bottom. The second data set is stacked on top of the first one. The third data set is stacked on the top of the second one, and so on.

The following property is specific to a stacked displayer:

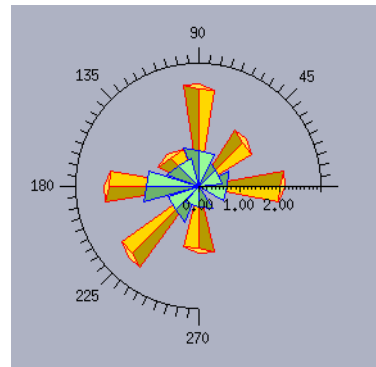
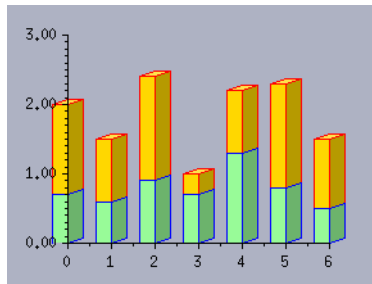
Property	Methods	Default Value
100% Stacked	<code>isStacked100PerCent</code> <code>setStacked100PerCent</code>	<code>IlvFalse</code>

Figure 8.15 illustrates the fact that a stacked displayer can be used with all types of projections. Data sets are represented by a stacked bar displayer, a stacked 3D bar displayer, and a stacked polygon displayer in Cartesian charts (using a Cartesian projection) and in polar charts (using a polar projection).

Stacked Bar Displayer



Stacked 3D Bar Displayer



Stacked Polygon Displayer

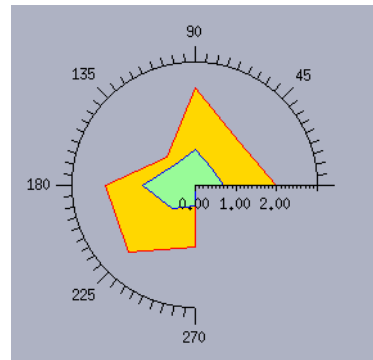
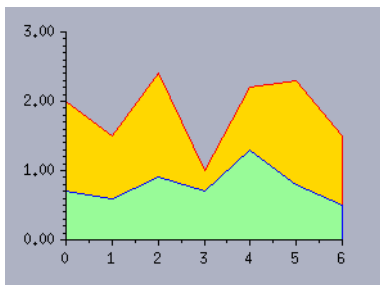


Figure 8.15 Stacked Displayers in Cartesian Charts and in Polar Charts

We can use the stacked displayer displaying data with polygons as an example. This displayer can be created in one of the following ways:

- ◆ By using the `IlvStackedPolygonChartDisplayer` class.
- ◆ Directly from the `IlvStackedChartDisplayer` class by specifying a displayer model.

- ◆ Directly from the `IlvStackedChartDisplayer` class by specifying a displayer factory.

The complete source code for creating the Cartesian chart displaying stacked data with stacked polygons can be found in the `stackedpolygon.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory. This file contains the code for each of the three ways of creating a stacked displayer.

Using a Subclass of the `IlvStackedChartDisplayer` Class

To create the stacked displayer by using a subclass of the `IlvStackedChartDisplayer` class, we can use the following code:

```
IlvStackedChartDisplayer* stackedDisplayer =
    new IlvStackedPolygonChartDisplayer(2, palettes);
```

Using a Displayer Model

To create the stacked displayer by specifying a displayer model, we can use the following code:

```
IlvAbstractChartDisplayer* model = new IlvPolygonChartDisplayer();
IlvStackedChartDisplayer* stackedDisplayer =
    new IlvStackedChartDisplayer(model, 2, palettes);
```

Using a Displayer Factory

To create the stacked displayer by specifying a displayer factory, we can use the following code:

```
PolygonDisplayerFactory* factory = new PolygonDisplayerFactory();
stackedDisplayer = new IlvStackedChartDisplayer(factory, 2, palettes);
```

The following code defines the displayer factory:

```
class PolygonDisplayerFactory
    : public IlvChartDisplayerFactory
{
public:

    PolygonDisplayerFactory()
        : IlvChartDisplayerFactory() {}

    virtual IlvChartDisplayerFactory* copy() const;

    virtual IlvAbstractChartDisplayer*
        createDisplayer (IlvCompositeChartDisplayer* parent,
                        IlvUInt idx,
                        IlvPalette *palette);
};

IlvChartDisplayerFactory*
PolygonDisplayerFactory::copy() const
{
    return new PolygonDisplayerFactory();
}

IlvAbstractChartDisplayer*
PolygonDisplayerFactory::
createDisplayer (IlvCompositeChartDisplayer* /* parent */,
                IlvUInt /* idx */,
                IlvPalette * palette)
{
    return new IlvPolygonChartDisplayer(palette);
}
```

Side-by-Side Displayers

The base class used to represent data side by side is the `IlvSideBySideChartDisplayer` class. The type of the child displayers that are created by default to be displayed is not specified at that level of the class hierarchy. However, it is possible to define the child displayers that will be created by setting one of the following:


- ◆ A displayer model
- ◆ A displayer factory

See *Using Composite Displayers* on page 171 for details on these two objects.

The Charts Library provides one subclass of the `IlvSideBySideChartDisplayer` class that defines the child displayers that are created by default:

`IlvSideBySideBarChartDisplayer` that displays the data as side-by-side bars.

A side-by-side displayer has the following basic characteristics:

Class	<code>IlvSideBySideChartDisplayer</code>
Category	Composite
Number of real data sets visualized	As many as you want
Can be used with all types of projections	Yes
Number of child displayers	As many as the number of real data sets
Subclasses	<code>IlvSideBySideBarChartDisplayer</code> 

A side-by-side displayer displays as many data sets as you want. Within the graphical representation, the data items of the first data set appear first. The data items of the second data set then appear next to the data items of the first data set. The data items of the third data set then appear next to the second data set, and so on.

Figure 8.16 illustrates the fact that a side-by-side displayer can be used with all types of projections. Data sets are represented by a side-by-side bar displayer in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

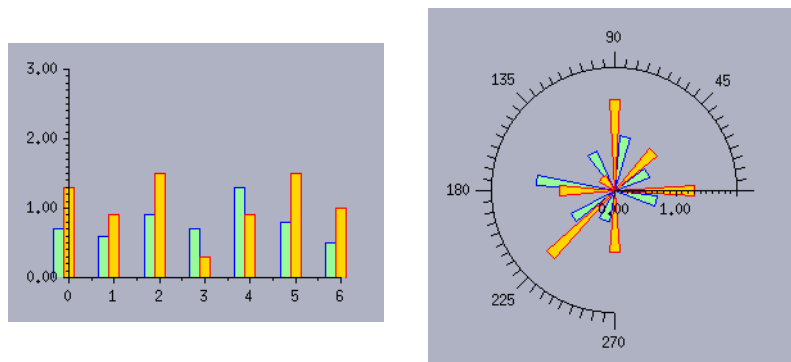


Figure 8.16 Side-by-Side Bar Displays in a Cartesian Chart and in a Polar Chart

We can use the side-by-side bar displayer as an example. We can easily create this displayer by using the `IlvSideBySideBarChartDisplayer` class.

```
IlvPalette* palettes[2];
palettes[0] = dpy->getPalette(dpy->getColor("palegreen"),
                             dpy->getColor("blue"));
palettes[1] = dpy->getPalette(dpy->getColor("gold"),
                             dpy->getColor("red"));
IlvSideBySideChartDisplayer* sideBySideDisplayer =
    new IlvSideBySideBarChartDisplayer(2, palettes);
```

You can also create a side-by-side displayer directly from the `IlvSideBySideChartDisplayer` class by specifying a displayer model or a displayer factory. See *Stacked Displayers* on page 177 to see how to create a displayer by specifying a displayer model or a displayer factory.

Adding a Displayer to a Chart

Once a displayer has been created, it should be added to the chart that needs to use it to display data.

The following methods are available to add a displayer:

```
IlvBoolean addDisplayer(IlvAbstractChartDisplayer* displayer,
                       IlvChartDataSet* dataSet = 0,
                       IlvChartCoordinateInfo* ordinateInfo = 0,
                       IlvUInt position = IlvLastPositionIndex)
```

and

```
virtual IlvBoolean addDisplayer(IlvAbstractChartDisplayer* displayer,
                                IlvUInt count,
                                IlvChartDataSet* const* dataSets,
                                IlvChartCoordinateInfo* ordinateInfo = 0,
                                IlvUInt position = IlvLastPositionIndex)
```

The first method is used for adding displayers that display only one data set, such as the scatter displayer, the polyline displayer, the marked polyline displayer, and so on. The second method is used for adding displayers that display several data sets, such as the high-low displayer, the high-low open-close displayer, and so on.

The data set(s) that will be displayed by the added displayer should be passed as a parameter. Otherwise, the displayer will be added but the corresponding graphical representation will not be displayed since no data to be represented are specified.

Note: *The data sets that are passed as parameters must be managed by the chart data object set on the chart.*

It is also possible to specify the ordinate scale that will be considered by the displayer to display the data. This is done by passing the coordinate information object associated with the ordinate scale as a parameter. If no coordinate information object is passed as a parameter, the ordinate scale that will be considered is the main ordinate scale. If several ordinate scales are displayed, the main ordinate scale corresponds to the ordinate scale displayed by the first ordinate scale displayer object that is defined (that is, the ordinate scale displayer at the index 0).

You can also specify the position at which the displayer is added. This allows you to indicate the order in which you want the graphical representations of data to be displayed since the displayers are considered in the order of the indexes.

Examples

We can use a scatter displayer that displays a single data set as an example. We can add this displayer to a chart by using the following code:

```
chart->addDisplayer(scatterDisplayer, dataSet);
```

See *Scatter Displayer* on page 155 to see how to create a scatter displayer.

Another example is a high-low open-close displayer that displays four data sets. We can add this displayer to a chart by using the following code:

```
IlvChartDataSet* dataSet[4];
dataSets[0] = lowValuesDataSet;
dataSets[1] = highValuesDataSet;
dataSets[2] = openValuesDataSet;
dataSets[3] = closeValuesDataSet;

chart->addDisplayer(hiLoDisplayer, 4, dataSet);
```

See *High-Low Open-Close Displayer* on page 175 to see how to create a high-low open-close displayer.

Customizing Data Display

By default, data are simply displayed by the defined displayers using the palette(s) set on the displayers. However, you can customize your data display as follows:

- ◆ Display an annotation linked to a given data point.
- ◆ Display a given data point using a specific palette.
- ◆ Projecting out-of-bounds data points onto the limits of the data display area (available for only a few displayers).

Adding Graphic Information to a Data Point

You can add specific graphic information to a data point of a given data set. This information is stored in a dedicated object called a *point information* object. The information will be treated at the time the chart is drawn.

A point information object is an instance of the `IlvChartDataPointInfo` class or one of its derived classes.

- ◆ The `IlvChartDataPointInfo` class allows you to draw the data point with a specific palette.
- ◆ The `IlvChartDataGraphicInfo` class allows you to draw any kind of graphic object next to the graphic representation of the data point. This graphic object can be used to define an annotation, put a marker on a given data point, and so on.
- ◆ The `IlvChartGradientPointInfo` class allows you to define a color gradient according to which data points will be drawn. The color of the point will be according to its value.

The point information objects defined for the data points of a given data set are managed by a dedicated object called a *point information collection* object. A point information object can be shared among different data points and a point information collection object can be shared among several data sets.

A point information collection object is represented by an instance of a subclass of the `IlvPointInfoCollection` class. Three subclasses are predefined:

- ◆ `IlvPointInfoMap`

This class stores in arrays both the point information objects defined for the data points of a given data set and the indexes of the data points with which the point information objects are associated. The stored indexes are used to retrieve the point information object associated with a given data point.

- ◆ `IlvPointInfoArray`

This class stores in an array the point information objects defined for the data points of a given data set. The point information object associated with the data point at a given index in a data set is stored at the same index in the `IlvPointInfoArray` object that is associated with the data set.

Note: *It is better to use an `IlvPointInfoMap` object rather than an `IlvPointInfoArray` object if you do not want to associate a point information object with each data point of a given data set.*

- ◆ `IlvPointInfoSingleton`

This class stores a unique point information object. It allows to associate the same point information object to all the data points of a data set.

A point information object can be associated with a data point at the level of a data set or at the level of a displayer. If the point information object is associated at the level of the data set, the point information object will be taken into account for each graphical representation of the data set that is performed in a chart. If the point information object is associated at the level of a displayer, it will be taken into account only in the graphical representation displayed by the displayer.

Notes:

1. *Before adding a point information object to a data point at the level of a data set, you should set a point information collection object on the data set.*
2. *Before adding a point information object to a data point at the level of a displayer, you should set a point information collection object on the displayer.*
3. *If two point information objects are associated with a given data point (one at the level of the data set and one at the level of a displayer displaying the data set), only the point information object associated at the level of the displayer will be displayed by this displayer.*

Example

We can use the Temperatures Chart that was introduced in *General Architecture of the Charts Library* on page 99 as an example. Two sets of data representing the morning and afternoon mean temperatures are represented in different ways on the two charts. Figure 8.17 shows these two charts.

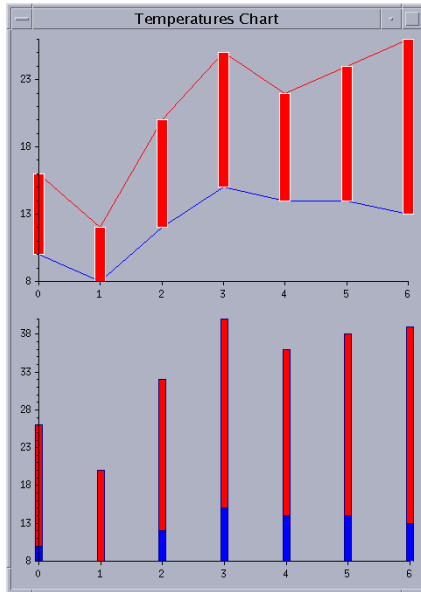


Figure 8.17 Two Versions of the Temperatures Chart

We want to customize how the data point corresponding to the highest afternoon temperature is displayed. We are going to add an annotation to the data point and display the data point with a specific palette. To do this, we need to add a point information object of the `IlvChartDataGraphicInfo` class to this data point.

The complete source code for this example can be found in the `pointinfo.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory. The following sections describe how to create and add the point information object to the data point.

Creating the Point Information Object

We want to display the text “Highest Afternoon Temperature” next to the data point corresponding to the highest afternoon temperature. We also want to display this data point with a specific palette. To do this, we create a point information object that stores an `IlvLabel` object and the specific palette for displaying the data point.

```
IlvChartDataPointInfo* pointInfo =
    new IlvChartDataGraphicInfo(new IlvLabel(display, 0, 0,
                                             "Highest afternoon temperature"),
                                0, 0, IlvTopRight,
                                display->getPalette(display->getColor("green"),
                                                       display->getColor("white")));
```

The defined label will be drawn with its top right corner located at the offset (0, 0) from the data point with which the point information object will be associated.

Adding the Point Information Object to the Data Point

Next, we need to associate the point information object with the data point corresponding to the highest afternoon temperature. This data point is at the index 6 in the afternoon mean temperatures data set. We can add the point information object either at the level of the data set or at the level of the displayer.

- ◆ To add the point information object at the level of the data set, use the following code:

```
// Set the point information collection object.
dataSets[1]->setPointInfoCollection(new IlvPointInfoMap());

// Set the point information object.
dataSets[1]->setPointInfo(6, pointInfo);
```

where `dataSets[1]` corresponds to the afternoon mean temperatures data set.

The last line of code is equivalent to:

```
dataSets[1]->getPointInfoCollection()->setPointInfo(6, pointInfo);
```

- ◆ To add the point information object at the level of a displayer displaying the data set, use the following code:

```
// Set the point information collection object.
chart1->getDisplayer(2)->setPointInfoCollection(dataSets[1],
                                                new IlvPointInfoMap());

// Set the point information object.
chart1->getDisplayer(2)->getPointInfoCollection(dataSets[1])
    ->setPointInfo(6, pointInfo);
```

where `chart1->getDisplayer(2)` corresponds to the high-low bar displayer representing the afternoon mean temperatures data set in the first chart.

The two charts obtained once the point information has been added from these examples are shown in the following figures. Figure 8.18 shows the charts obtained when the point information object is added at the level of the data set. Figure 8.19 shows the charts obtained when the point information object is added at the level of the displayer.

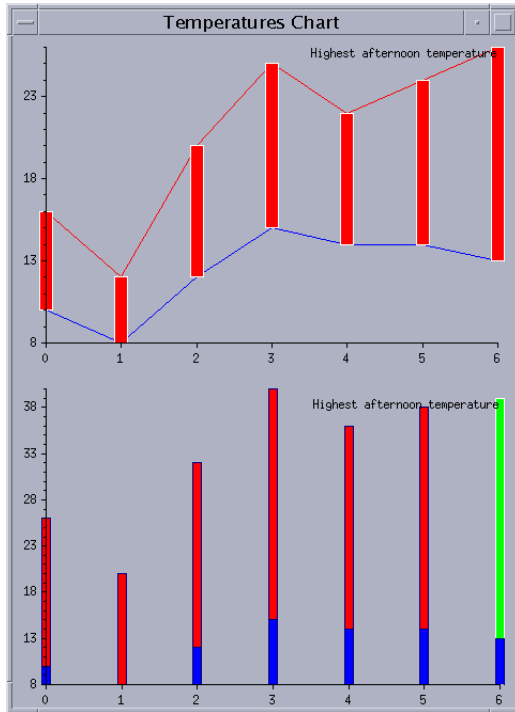


Figure 8.18 Point Information Object Associated with a Data Point at the Level of a Data Set

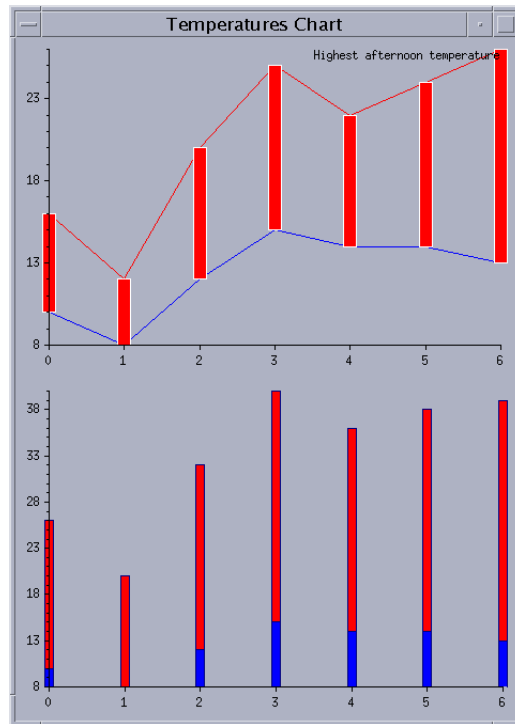


Figure 8.19 Point Information Object Associated with a Data Point at the Level of a Displayer

You can see that when the point information object is associated at the level of the data set (Figure 8.18), the stored label is displayed for all the graphical representations of the data set. Similarly, the stored specific palette is used to display the data point by all the displayers displaying the data set.

On the other hand, when the point information object is associated at the level of a displayer (Figure 8.19), the stored label is displayed only by the displayer. Similarly, the stored specific palette is used to display the data point only by the displayer.

Defining How the Palettes are Applied for the Data Display

By default, the palette used by a displayer to display a data point of a data set is determined in the following order:

- ◆ The palette stored in the point information object associated with the data point at the level of the displayer is used if such a palette is defined.
- ◆ Otherwise, the palette stored in the point information object associated with the data point at the level of the data set is used if such a palette is defined.

- ◆ Otherwise, the palette defined for the displayer is used if such a palette is defined.
- ◆ Otherwise, the palette defined for the chart using the displayer is used if such a palette is defined.
- ◆ Otherwise, the default palette of the display is used.

If a graphical representation by data point for the considered displayer exists, the palette is directly used to draw the graphical representation associated with the data point. This is the case for the displayers displaying markers, bars, and so on. The Boolean value indicating whether there is a graphical representation by data point for a given displayer is returned by the method

```
IlvAbstractChartDisplayer::graphicalRepresentationByDataPoint.
```

For some displayers (such as polyline displayers, step displayers, and so on) no graphical representation by data point exists. These displayers display a continuous graphical representation linking several data points. For such displayers, the palette is applied by default after the data point. However, it is possible to apply the palette before the data point by calling the method `IlvChartGraphic::setApplyPaletteAfterPoint` of the chart with `IlvFalse` as a parameter.

Using the Predefined `IlvChartGradientPointInfo` Class

You can get the data points to be displayed in color gradations which are computed according to the point value. To do this, you need to build an `IlvChartGradientPointInfo` with an `IArray` of colors and an array of double values. There must be the same number of colors and values. The first value should be the minimum data range and the last one should be the maximum data range.

Example:

```
Ildouble values[ ] = {0, 50, 100};
IArray colors;
IlvColor *col = display->getColor("blue");
colors.add(col);
col = display->getColor("white");
colors.add(col);
col = display->getColor("red");
colors.add(col);

gradientInfo = new IlvChartGradientPointInfo(values, colors);
```

The color of each data point which is between 0 and 100 will be computed according to the passed colors. 0 is pure blue, 50 is pure white, 100 is pure red. intermediate values will be displayed in a computed gradient of these colors.

Projecting Out-of-Bounds Data Points

By default, the data points that should appear outside of the data display area (if such data points exist) are not drawn. The graphical representations of such data points are simply

clipped by the data display area. However, for the displayers that are instances of the `IlvPolylineChartDisplayer` class or one of its subclasses, it is possible to display the out-of-bounds data points projected on the limits of the data display area. For details, see *Polyline Displayer* on page 156.

Scales Display

This chapter provides detailed information on displaying the scales of charts. You can find information on the following topics:

- ◆ *Drawing the Scales of a Chart*
- ◆ *Using Single Scale Displayers*
- ◆ *Using Scale Steps Updaters to Compute Scales Graduations*
- ◆ *Adding a Scale Displayer in a Chart*
- ◆ *Advanced Features for Customizing Scales*

Drawing the Scales of a Chart

A scale is displayed within a chart by a dedicated object called a *scale displayer*.

The base class used to represent a scale displayer is the `IlvAbstractScaleDisplayer` class.

A subclass is provided called `IlvSingleScaleDisplayer`. This class represents a *single* coordinate on an axis.

The *single* scales representing several ordinate coordinates can be stacked in order to represent them on the same axis.

Figure 9.1 shows the hierarchy of all the scale displayers defined in the Charts Library.

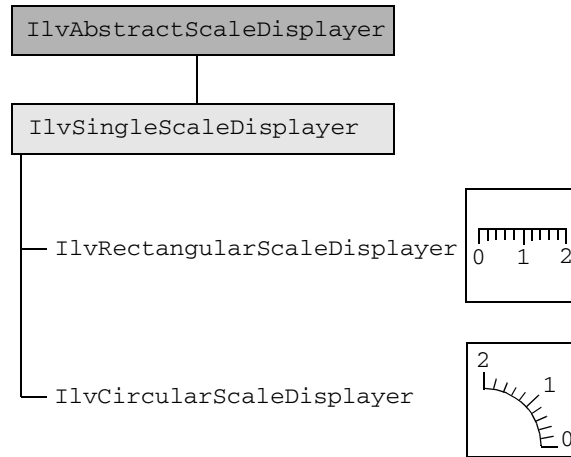


Figure 9.1 Hierarchy of Scale Displayers in the Charts Library

Setting General Properties

The following properties are defined for all the scale displayers:

Property	Methods	Default Value
Scale Positioning Properties		
Fixed to a Position	isFixedToPosition getRelativePosition setRelativePosition	IlvTrue IlvMinDataPosition
Fixed to a Data Value	getCrossingValue setCrossingValue	0
Scale Display Properties		
Visibility	isVisible setVisible	IlvTrue
Drawing Order Relative to the Drawing of the Graphical Representations of Data	getDrawOrder setDrawOrder	IlvDrawAbove
Must Always Appear in the Data Display Area	isAlwaysVisible setAlwaysVisible	IlvFalse

Property	Methods	Default Value
Miscellaneous		
Name	getName setName	0
Flags	getFlags setFlags	0

Defining the Position of a Scale

A scale can be fixed either to a position or to a data value:

◆ Fixing a scale to a position

This can be done by means of the

`IlvAbstractScaleDisplayer::setRelativePosition` method. The position at which a scale is fixed is defined by an offset from the position at which the minimum or the maximum data value is displayed.

For example, the following code line fixes the position of the scale to an offset of -10 pixels from the position at which the minimum data value is displayed:

```
scaleDisplayer->setRelativePosition(IlvMinDataPosition, -10);
```

The following code line fixes the position of the scale to the position at which the maximum data value is displayed:

```
scaleDisplayer->setRelativePosition(IlvMaxDataPosition);
```

◆ Fixing a scale to a data value of another scale

This can be done by means of the

`IlvAbstractScaleDisplayer::setCrossingValue` method. You have to specify the scale and the data value of this scale to which you want to fix the position of the current scale. If the current scale is the abscissa scale, the other scale must be an ordinate scale. If the current scale is an ordinate scale, the other scale must be the abscissa scale.

The following code line fixes the position of the scale displayed by the scale displayer `scaleDisplayer` to the data value 2.0 on the scale displayed by the scale displayer `otherScaleDisplayer`.

```
scaleDisplayer->setCrossingValue(2.0, otherScaleDisplayer);
```

If a scale is fixed to a data value that does not belong to the range of visible data, the scale will not appear on the screen. However, if the property “Must Always Appear in

the Data Display Area” is set to `ILVTRUE`, the range of visible data will be automatically modified so that the scale can appear on the screen.

Note: By default, a scale is fixed to the position at which the minimum data value is displayed.

Defining the Drawing Order Relative to the Drawing of the Graphical Representations of Data

A scale can be displayed on top of (`ILVDrawAbove`) or underneath (`ILVDrawBelow`) the graphical representations of data in a chart. By default, a scale will be displayed on top of the graphical representations. However, the drawing order can be changed for a given scale by means of the `ILVAbstractScaleDisplayer::setDrawOrder` method.

Using Single Scale Displayers

The base class used to represent a single scale is the `ILVSingleScaleDisplayer` class.

A single scale is composed of:

- ◆ An *axis*, which can have an optional *arrow* and an optional *label* at the end
- ◆ *Major ticks*, the marks drawn on the axis at each *step* of the scale
- ◆ *Step labels* drawn next to the major ticks. These labels indicate the values of the coordinate represented by the scale.
- ◆ *Minor ticks*, the marks drawn on the axis at each *substep* of the scale

Figure 9.2 shows the different elements of a single scale.

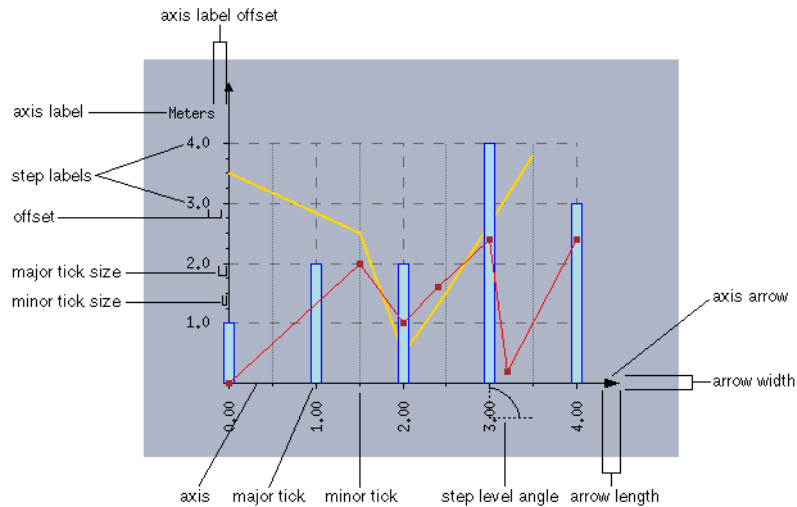


Figure 9.2 Elements of a Single Scale

Setting General Properties

The following properties are defined for all the single scale displayers. The methods followed by an asterisk (*) are defined as pure virtual at the level of the `IlvAbstractScaleDisplayer` class and are redefined at the level of the `IlvSingleScaleDisplayer` class.

Table 9.1

Property	Methods	Default Value
Step and Label Definition Properties		
Scale Steps Updater	<code>getStepsUpdater</code>	<code>IlvAutoScaleStepsUpdater</code>
Step and Substep Numbers	<code>getStepsCount</code>	0
	<code>getSubStepsCount</code>	0
	<code>getTotalSubStepsCount</code>	0
Step and Substep Units	<code>getStepUnit</code>	0
	<code>getSubStepUnit</code>	0
Definition of Text Labels by Hand	<code>getStepLabelsCount</code>	0
	<code>getStepLabel</code>	
	<code>getStepLabels</code>	
	<code>setStepLabel *</code>	
	<code>setStepLabels *</code>	

Table 9.1

Property	Methods	Default Value
Step Labels Computation: By Applying a Format	getStepLabelFormat setStepLabelFormat *	IlvDefaultStepLabelFormat
By Applying a Callback Converting Data Values into Step Labels	getValueToLabelCB getValueToLabelCBData setValueToLabelCB *	0 0
Ticks and Labels Display Properties		
Layout Properties: Position of Ticks	getTickLayout setTickLayout *	TickOutside
Position of Labels	getLabelLayout setLabelLayout *	LabelOutside
Offset Between Ticks and Step Labels	getOffset setOffset *	IlvDefaultScaleOffset
Size Properties: Major Ticks	getMajorTickSize setMajorTickSize *	IlvDefaultScaleMajorTickSize
Minor Ticks	getMinorTickSize setMinorTickSize *	IlvDefaultScaleMinorTickSize
Step Labels	getStepLabelSizes	
Step Label Properties: Angle	getStepLabelAngle setStepLabelAngle *	0
Palette	getStepLabelsPalette setStepLabelsPalette *	0
Drawn at Axes Crossings	isDrawingLabelOnCrossings drawLabelOnCrossings *	IlvFalse
Drawn When Overlapping	isDrawingOverlappingLabels drawOverlappingLabels *	IlvTrue

Table 9.1

Property	Methods	Default Value
Visibility: Major Ticks	areMajorTicksVisible setMajorTicksVisible	IlvTrue
Minor Ticks	areMinorTicksVisible setMinorTicksVisible	IlvTrue
Step Labels	areStepLabelsVisible setStepLabelsVisible	IlvTrue
Axis Display Properties		
Arrow at the End: Arrow Drawn	isAxisOriented setAxisOriented *	IlvFalse
Arrow Width	getArrowWidth setArrowWidth *	IlvDefaultScaleArrowWidth
Arrow Length	getArrowLength setArrowLength *	IlvDefaultScaleArrowLength
Axis Label: Label	getAxisLabel setAxisLabel	0
Offset Between Axis and Label	getAxisLabelOffset setAxisLabelOffset *	2*IlvDefaultScaleOffset
Size	getAxisLabelSizes	
Palettes: Axis	getAxisPalette setAxisPalette *	0
Axis Label	getAxisLabelPalette setAxisLabelPalette *	0
Visibility	isAxisVisible setAxisVisible	IlvTrue

Defining the Steps and Substeps

The computation of the steps and substeps for a given scale is performed by a dedicated object called *scale steps updater* that is set on the scale. This object is returned by the `IlvSingleScaleDisplayer::getStepsUpdater` method.

For more details on the scale steps updaters, see the section *Using Scale Steps Updaters to Compute Scales Graduations*.

Defining the Step Labels to be Displayed

By default, the step labels that are displayed are simply numerical labels corresponding to the data values represented by the scale. These labels are formatted by applying the format returned by the `IlvSingleScaleDisplayer::getStepLabelFormat` method.

However, you can change the step labels to be displayed by default by doing one of the following:

- ◆ By specifying text labels by hand

This can be done by means of the `IlvAbstractScaleDisplayer::setStepLabels` method. The labels that will be displayed are then the text labels set by hand..

Note: *In this case, the number of steps will be equal to the number of text labels that are set by hand.*

- ◆ By defining a callback indicating how to convert a given data value into a step label

This callback must be of the `IlvValueToLabelCB` type:

```
typedef char* (* IlvValueToLabelCB )(IlDouble, IlAny);
```

This kind of callback can be set by means of the

`IlvAbstractScaleDisplayer::setValueToLabelCB` method. The labels that will be displayed are those returned by the callback that is set.

Example: Callback Converting Data Values into Step Labels

The following example shows how to define a callback that displays data values expressed in seconds in the form “hours-minutes-seconds.”

1. To define the callback, use the following code:

```
char*
hours_minutes_seconds(IlDouble value, IlAny cbData)
{
    char buffer[126];
    IlvUInt modulo;
    IlvUInt hours;
    IlvUInt minutes;
    IlvUInt seconds;
    if (value >= 3600) {
        hours = (IlvUInt)value / 3600;
        modulo = (IlvUInt)value % 3600;
    }
    else {
        modulo = value;
        hours = 0;
    }
    if (modulo >= 60) {
        minutes = modulo / 60;
        seconds = modulo % 60;
    }
    else {
        minutes = 0;
        seconds = modulo;
    }
    sprintf(buffer, "%d-%d-%d", hours, minutes, seconds);
    return IlvCopyString(buffer);
}
```

2. To set the callback on the scale displayer, use the following code:

```
scaleDisplayer->setValueToLabelCB(hours_minutes_seconds);
```

Defining the Position of the Ticks Relative to the Axis

The position of the ticks relative to the axis is defined by the enumeration type `TickLayout`. The following positions are possible:

- ◆ `TickInside`

The ticks extend inside of the data display area.

- ◆ `TickOutside`

The ticks extend outside of the data display area.

- ◆ `TickCross`

The ticks cross the axis and extend both inside and outside of the data display area.

By default, the position of the ticks of a scale is set to `TickOutside`. This setting can be changed by means of the `IlvAbstractScaleDisplayer::setTickLayout` method.

Defining the Position of the Labels Relative to the Axis

The position of the labels relative to the axis is defined by the enumeration type `LabelLayout`. The following positions are possible:

- ◆ `LabelInside`

The labels extend inside of the data display area.

- ◆ `LabelOutside`

The labels extend outside of the data display area.

By default, the position of the labels of a scale are set to `LabelOutside`. This position applies both to the step labels and to the axis label. The setting can be changed by means of the `IlvAbstractScaleDisplayer::setLabelLayout` method.

Defining Whether the Step Labels are Drawn at Axes Crossings

By default, step labels are not drawn at the axes crossings. However, you can specify that the step labels should be drawn at the axes crossings for a given scale by calling the `IlvAbstractScaleDisplayer::drawLabelOnCrossings` method with `IlvTrue` as a parameter.

Defining Whether Overlapping Step Labels are Drawn

By default, step labels are drawn even if they overlap. However, you can specify that overlapping step labels should not be drawn for a given scale by calling the `IlvAbstractScaleDisplayer::drawOverlappingLabels` method with `IlvFalse` as a parameter.

Predefined Single Scale Displayers

The following sections describe the single scale displayers of the Charts Library. For each scale displayer, you will find a table that describes the conditions for using the scale displayer, including whether the scale displayer can be used whatever the type of the projection is and for which coordinate the scale displayer can be used.

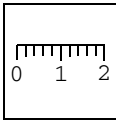
You can find information on the following single scale displayers:

- ◆ *Rectangular Scale Displayer*

- ◆ *Circular Scale Displayer*

Rectangular Scale Displayer

A *rectangular scale* is a scale that has a straight line as an axis. A rectangular scale displayer allows you to display a rectangular scale. This type of scale displayer has the following basic characteristics:

Class	<code>IlvRectangularScaleDisplayer</code>
Category	Single
Can be used with all types of projections	Yes
Use	Used to display the scale representing: <ul style="list-style-type: none"> - The abscissa and ordinate coordinates in Cartesian charts. - The ordinate coordinates in polar charts.
Shape	Rectangular 

The way the rectangular scales are positioned depends on the type of the projection. In Cartesian charts (using a Cartesian projection), they will be displayed orthogonally. In polar charts (using a polar projection), they will be displayed radially.

Figure 9.3 shows rectangular scales displayed by rectangular scale displayers in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

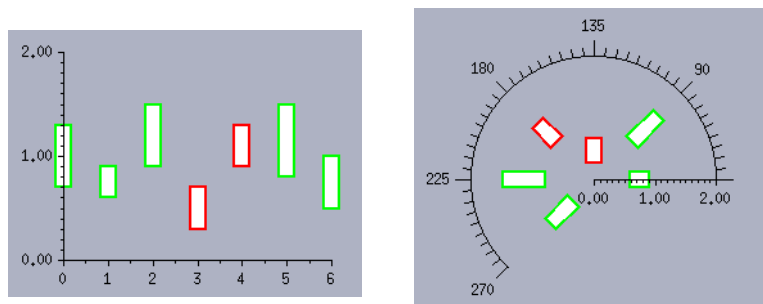


Figure 9.3 *Rectangular Scales Displayed by Rectangular Scale Displayers*

We can use the rectangular scale that represents the ordinate coordinate in both charts as an example. We can create the scale displayer displaying this scale by using the following code:

```

IlvCoordinateInfo* coordInfo =
    new IlvCoordinateInfo(IlvOrdinateCoordinate);
IlvRectangularScaleDisplayer* ordinateScaleDisplayer =
    new IlvRectangularScaleDisplayer(coordInfo,
                                     chart->getPalette());
ordinateScaleDisplayer->drawLabelOnCrossings(IlvTrue);
IlvConstantScaleStepsUpdater* updater =
    new IlvConstantScaleStepsUpdater(ordinateScaleDisplayer);
delete IlvScaleStepsUpdater::Set(ordinateScaleDisplayer, updater);
updater->fixStepsCount(1, 0.1);

```

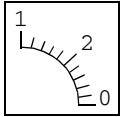
The coordinate information object associated with the coordinate that is represented by the scale is passed as a parameter to the constructor of the scale displayer. The palette that will be used to display the scale is set to the default palette of the chart.

The Boolean value indicating whether the step labels are drawn at the axes crossings is set to `IlvTrue`.

Then a *constant* scale steps updater is set on the scale, since we want steps and substeps with constant spacing, and finally we set a step unit of 1 and a substep unit of 0.1 on the scale steps updater.

Circular Scale Displayer

A *circular scale* is a scale that has a portion of a circle as an axis. A circular scale displayer allows you to display a circular scale. This type of scale displayer has the following basic characteristics:

Class	<code>IlvCircularScaleDisplayer</code>
Category	Single
Can be used with all types of projections	No (Use only with a polar projection)
Use	Used to display the scale representing: - The abscissa coordinate in polar charts
Shape	Circular 

A circular scale can be used only to represent the abscissa coordinate in polar charts. Figure 9.4 shows a circular scale displayed by a circular scale displayer in a polar chart (using a polar projection).

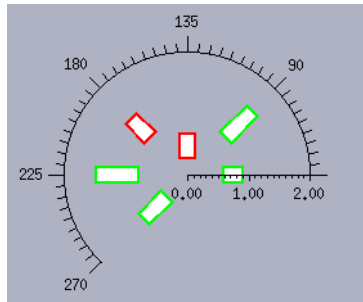


Figure 9.4 Circular Scale Displayed by a Circular Scale Displayer

We can use the circular scale representing the abscissa coordinate in the above chart as an example. We can create the scale displayer displaying this scale by using the following code:

```
IlvCoordinateInfo* coordInfo =
    new IlvCoordinateInfo(IlvAbscissaCoordinate);
IlvCircularScaleDisplayer* abscissaScaleDisplayer =
    new IlvCircularScaleDisplayer(coordInfo,
        chart->getPalette());
abscissaScaleDisplayer->setMajorTickSize(12);
abscissaScaleDisplayer->setMinorTickSize(6);
abscissaScaleDisplayer->setStepLabelFormat("%.0f");
IlvConstantScaleStepsUpdater* updater =
    new IlvConstantScaleStepsUpdater(abscissaScaleDisplayer);
delete IlvScaleStepsUpdater::Set(abscissaScaleDisplayer, updater);
updater->fixStepUnit(45.,5.);
```

The coordinate information object associated with the coordinate that is represented by the scale is passed as a parameter to the constructor of the scale displayer. The palette that will be used to display the scale is set to the default palette of the chart.

The size of the steps is set to 12 and the size of the substeps is set to 6. The format of the step labels is set to "%.0f". This means that the step labels will be drawn in the form of integer values.

Then a *constant* scale steps updater is set on the scale, since we want steps and substeps with constant spacing, and finally we set a step unit of 45 and a substep unit of 5 on the scale steps updater.

Using Scale Steps Updaters to Compute Scales Graduations

The computation of the steps and substeps is performed by a dedicated object called *scale steps updater* that is set on the scale.

The base class used to represent a scale steps updater is the `IlvScaleStepsUpdater` class. Several subclasses are provided in the Charts Library:

◆ `IlvConstantScaleStepsUpdater`

This class allows you to define graduations with constant spacing.

The steps and the substeps can be defined by:

- Fixing the number of steps and substeps. This can be done by means of the `IlvConstantScaleStepsUpdater::fixStepsCount` method.
- Fixing the step and substep units. This can be done by means of the `IlvConstantScaleStepsUpdater::fixStepsCount` method.

In the case of a fixed number of steps and substeps, the first major tick of the scale is drawn for the minimum value of the data interval represented by the scale and the last major tick is drawn for the maximum value of the data interval represented by the scale. However, the data values associated with the first and the last steps can also be set by hand by means of the `IlvConstantScaleStepsUpdater::setFirstStepData` and the `IlvConstantScaleStepsUpdater::setLastStepData` methods respectively.

◆ `IlvAutoScaleStepsUpdater`

This class is a subclass of `IlvConstantScaleStepsUpdater` which enables automatic computation of the step and substeps.

This automatic computation can be customized in several ways:

- The precision can be automatic or it can be set by hand by means of the `IlvAutoScaleStepsUpdater::setPrecision` method.
- The precision base is set to 10 by default. It can be modified by means of the `IlvAutoScaleStepsUpdater::setPrecision` method.
- The step label format can be automatic or not.
- The substeps computing can be automatic or the number of substeps between two steps can be set by hand by means of the `IlvAutoScaleStepsUpdater::setAutoSubSteps` method.
- The steps spacing is set to 10 by default. It can be modified by means of the `IlvAutoScaleStepsUpdater::setStepsSpacing` method.

◆ `IlvLogScaleStepsUpdater`

This class allows you to compute logarithmic graduations.

Warning: *This class can be used to compute the steps for a given scale only when a logarithmic transformation is set on the coordinate represented by the scale.*

◆ `IlvZoomScaleStepsUpdater`

This class allows you to display graduations that are zoomed locally on part of the scale, while the steps and substeps are displayed normally on the rest of the scale.

Warning: This class can be used to compute the steps for a given scale only if a transformer of the `IlvZoomCoordinateTransformer` type is also set on the coordinate represented by the scale.

You can look at the `lens.cpp` file located in the `$ILVHOME/samples/charts/lens/src` directory for an example of how to use both the `IlvZoomScaleStepsUpdater` and `IlvZoomCoordinateTransformer` classes.

◆ `IlvTimeScaleStepsUpdater`

This class allows you to compute graduations based on time units (seconds, minutes, hour, calendar dates, and so on).

You can either manually set which time unit to use (with `setTimeUnit()`, passing one of the predefined units in `charts/date.h`), or, like with `IlvAutoScaleStepsUpdater`, you can let it choose the "best" unit automatically, which is the default behavior. You can change this behavior with `IlvTimeScaleStepsUpdater::setAutoUnit()`.

Look at the `stock.cpp` file located in `$ILVHOME/samples/chart/interactors/src` directory for an example of how to use the `IlvTimeScaleStepsUpdater`.

Adding a Scale Displayer in a Chart

A chart uses one abscissa scale and as many ordinate scales as you want. The type of scales that can be used for a given chart depends on the type of the projection used in the chart.

The following table shows the scales that can be used with a Cartesian chart:

Chart type	Cartesian
Projection used	Cartesian -> <code>IlvCartesianProjector</code>
Scale Displayers:	
Abscissa Coordinate	Rectangular -> <code>IlvRectangularScaleDisplayer</code>
Ordinate Coordinate(s)	Rectangular -> <code>IlvRectangularScaleDisplayer</code>

The following table shows the scales that can be used with a polar chart:

Chart type	Polar
Projection used	Polar -> IlvPolarProjector
Scale Displayers:	
Abscissa Coordinate	Circular -> IlvCircularScaleDisplayer
Ordinate Coordinate(s)	Rectangular -> IlvRectangularScaleDisplayer

If the chart that you have created is an instance of the `IlvChartGraphic` class, no scale displayers are created by default.

If the chart that you have created is an instance of a subclass of the `IlvChartGraphic` class (that is, `IlvCartesianChart` or `IlvPolarChart`), two scale displayers are already created by default (one for the abscissa coordinate and one for the main ordinate coordinate):

- ◆ Two `IlvRectangularScaleDisplayer` objects are created by default for an `IlvCartesianChart` object.
- ◆ One `IlvCircularScaleDisplayer` object and one `IlvRectangularScaleDisplayer` object are created by default for an `IlvPolarChart` object.

The following method is available to set by hand the abscissa scale displayer of a given chart:

```
virtual void setAbscissaScale(IlvSingleScaleDisplayer* scale)
```

Similarly, several methods are available to set by hand the ordinate scale displayer(s) that you want to be used by a given chart.

- ◆ To add an ordinate scale displayer, use the following method:

```
void addOrdinateScale(IlvAbstractScaleDisplayer* scale)
```

The ordinate scale displayer is added at the end of the list of the ordinate scale displayers that are already defined, if any.

- ◆ To insert an ordinate scale at a given index in the list of ordinate scale displayers that are already defined, use the following method:

```
virtual void insertOrdinateScale(IlvAbstractScaleDisplayer* scale,
                                IlvUInt index = 0)
```

- ◆ To set the ordinate scale displayer at a given index in the list of ordinate scale displayers, use the following method:

```
virtual void setOrdinateScale(IlvUInt index,
                             IlvAbstractScaleDisplayer* scale)
```

If a scale displayer is already defined at this index, this scale displayer is replaced by the new one.

Note: When several ordinate scale displayers are defined for a given chart, the corresponding scales are drawn in the order of the indexes.

Advanced Features for Customizing Scales

Several of the features for customizing scales cannot be made by simply setting properties on scales. These advanced features require you to specify certain settings on other objects of the chart. These advanced features are the following:

- ◆ *Changing the Orientation of the Scales*
- ◆ *Defining the Minimum and Maximum Data Values Represented by a Scale*
- ◆ *Applying a Transformation to the Data Values Represented by a Scale*

Changing the Orientation of the Scales

The scales of a chart can have different orientations depending on the type of the projection used for the chart. You specify the orientation of the scales by means of the projector object, which is responsible for the projection of the data points into screen coordinates. This object is an instance of the `IlvCartesianProjector` class for Cartesian charts and an instance of the `IlvPolarProjector` class for polar charts.

Changing the Scale Orientation for Cartesian Charts

The orientation of the scales for Cartesian charts is defined by a value of the type `IlvCartesianProjector::Orientation` that is set on the Cartesian projector. Any orientation that keeps the abscissa and the ordinate scales orthogonal can be used. Figure 9.5 shows these possible orientations.

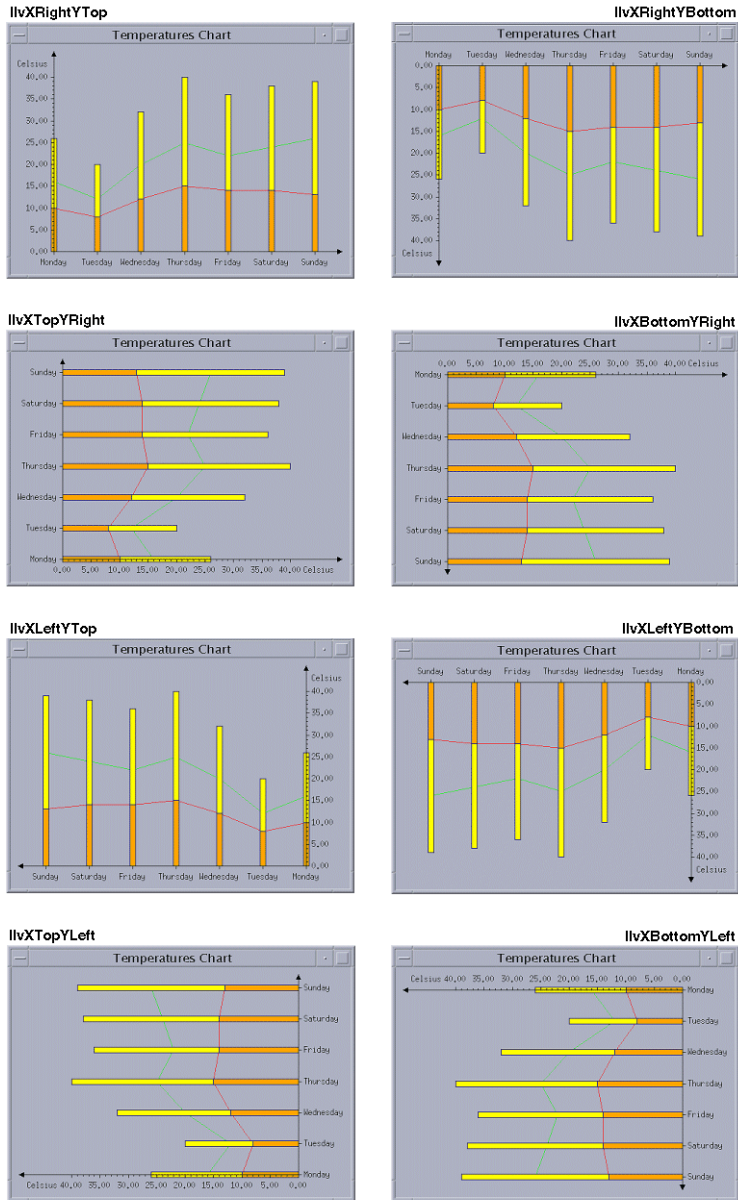


Figure 9.5 Scale Orientation in a Cartesian Chart

By default, the scales orientation is set to `IlvCartesianProjector::IlvXRightYTop`. This means that the abscissa scale is oriented towards the right of the screen and the ordinate

scales are oriented towards the top. To change the scales orientation, just call the `IlvCartesianProjector::setOrientation` method of the Cartesian projector used in your Cartesian chart. For example, to change the scales orientation so that the abscissa scale is oriented towards the right of the screen and the ordinate scale is oriented towards the bottom, you can use the following code:

```
IlvCartesianProjector* projector
    = (IlvCartesianProjector*) (chart->getProjector());
projector->setOrientation(IlvCartesianProjector::IlvXRightYBottom);
```

Changing the Scale Orientation for Polar Charts

The orientation of the abscissa scale for polar charts is determined by a flag at the level of the polar projector. This flag indicates whether the polar system of coordinates is oriented clockwise. The value of this flag is returned by the `IlvPolarProjector::getOrientedClockwise` method. By default, this flag is set to `IlvFalse`. This means that the abscissa scale will be oriented counterclockwise. You can change the orientation by means of the `IlvPolarProjector::setOrientedClockwise` method of the polar projector used in your polar chart. For example, to have the abscissa scale oriented clockwise, use the following code:

```
IlvPolarProjector* projector = (IlvPolarProjector*) (chart->getProjector());
projector->setOrientedClockwise(IlvTrue);
```

The orientation of the ordinate scale(s) for polar charts is not determined at the level of the projector. The orientation simply depends on the position of the ordinate scale(s) since they are displayed radially. For more information on how to position a scale, see *Defining the Position of a Scale* on page 196.

Defining the Minimum and Maximum Data Values Represented by a Scale

The minimum and maximum data values that are represented by a scale are stored in the coordinate information object associated with the coordinate represented by the scale.

This coordinate information object can be obtained by means of the `IlvAbstractScaleDisplayer::getCoordinateInfo` method of the scale displayer. It can also be accessed directly at the level of the chart without passing through the scale displayer. You can use the following methods to access the object at the level of the chart:

`IlvChartGraphic::getAbscissaInfo` for the abscissa coordinate

`IlvChartGraphic::getOrdinateInfo` for the ordinate coordinate(s)

By default, the minimum and maximum data values represented by a scale are computed automatically so that all the data points that are displayed by considering this scale can appear within the interval defined by the minimum and maximum data values. However, the minimum and maximum data values that are represented by a scale can be defined by using one of the following methods on the coordinate information object associated with the scale:

```

IlvCoordinateInfo::setUserDataMin
IlvCoordinateInfo::setUserDataMax
IlvCoordinateInfo::setUserDataRange

```

The minimum and maximum data values that are actually represented by a given scale, whether they are computed automatically or set by hand, are obtained by means of the following methods on the coordinate information object associated with the scale:

```

IlvCoordinateInfo::getDataMin
IlvCoordinateInfo::getDataMax
IlvCoordinateInfo::getDataRange

```

For example, the following code lines set 1 and 4 as the minimum and maximum data values represented by the main ordinate scale of a chart. This means that the data points that have to be displayed by considering this scale and that are outside of the interval defined by these minimum and maximum data values will not be displayed.

```

IlvAbstractScaleDisplayer* ordinateScale = chart->getOrdinateScale();
IlvChartCoordinateInfo* ordinateInfo = ordinateScale->getCoordinateInfo();
ordinateInfo->setUserDataRange(IlvCoordInterval(1, 4));

```

Applying a Transformation to the Data Values Represented by a Scale

Different types of transformations can be applied to the data values that are represented by a given scale. These transformations are defined by means of *transformer* objects that are all instances of subclasses of the `IlvChartCoordinateTransformer` class. Several subclasses are provided in the Charts Library:

◆ `IlvChartCoordinateTransformer`

This class is the base class permitting you to define transformations that consist of an elementary transformation, possibly followed by a logarithmic transformation.

The following subclasses are defined:

- `IlvSimpleChartTransformer` has an identity transformation as the elementary transformation. This means that no transformation will be applied before the logarithmic transformation (if a logarithmic transformation is defined).
- `IlvAffineChartTransformer` has an affine transformation as the elementary transformation. This means that an affine transformation will be applied before the logarithmic transformation (if a logarithmic transformation is defined).

The logarithmic transformation that is applied to the data values depends on the defined logarithmic base. If the logarithmic base is equal to 0 or 1, no logarithmic transformation is applied. Otherwise, the following transformation is applied:

$$c_transformed = \log(c) / \log(base)$$

where c is the initial coordinate, $base$ is the logarithmic base, log is the natural logarithm, and $c_{transformed}$ is the coordinate transformed using the logarithmic transformation.

The transformer object defining the transformation to be applied to a given coordinate is stored in the coordinate information object associated with the coordinate.

The following code lines show how to apply a logarithmic transformation to the data values represented along the abscissa of a chart.

```
IlvSingleScaleDisplayer* abscissaScale = chart->getAbscissaScale();
IlvChartCoordinateInfo* abscissaInfo = abscissaScale->getCoordinateInfo();
abscissaInfo->setTransformer(new IlvSimpleChartTransformer(10));
```

The base of the logarithmic transformation that will be applied is set to 10.

Warning: *If you want the abscissa scale to be graduated with logarithmic graduations, you must also set a logarithmic scale steps updater on the scale.*

To set a logarithmic scale steps updater on the abscissa scale use the following code:

```
IlvLogScaleStepsUpdater* logUpdater =
    new IlvLogScaleStepsUpdater(abscissaScale);
delete IlvScaleStepsUpdater::Set(abscissaScale, logUpdater);
```

◆ IlvZoomCoordinateTransformer

This class allows you to define a transformation that is applied locally on a part of the data to display that data as zoomed.

Warning: *If such a transformer is set on a given coordinate a zoom scale steps updater must also be set on the scale representing this coordinate in order to compute the corresponding graduations for the scale.*

You can look at the `lens.cpp` file located in the `$ILVHOME/samples/charts/lens/src` directory for an example of how to use the `IlvZoomCoordinateTransformer` and `IlvZoomScaleStepsUpdater` classes.

Decorations Display

The Charts Library provides the capability to add decorations to a chart that will improve its appearance or help in understanding the data. This chapter provides information on displaying these decorations in a chart. You will find information on the following topics:

- ◆ *Displaying a Legend*
- ◆ *Displaying a Grid*
- ◆ *Displaying a Cursor*

Displaying a Legend

The base class used to define a legend to be added to a given chart is the `IlvChartLegend` class. This class inherits from the `IlvShadowRectangle` class. Thus, the chart legend object is a graphic object just like the chart object and is positioned independently of the chart object.

Figure 10.1 shows an example of a legend that has been added to a chart. This legend is composed of three legend items and is surrounded by a filled rectangle with a shadow.

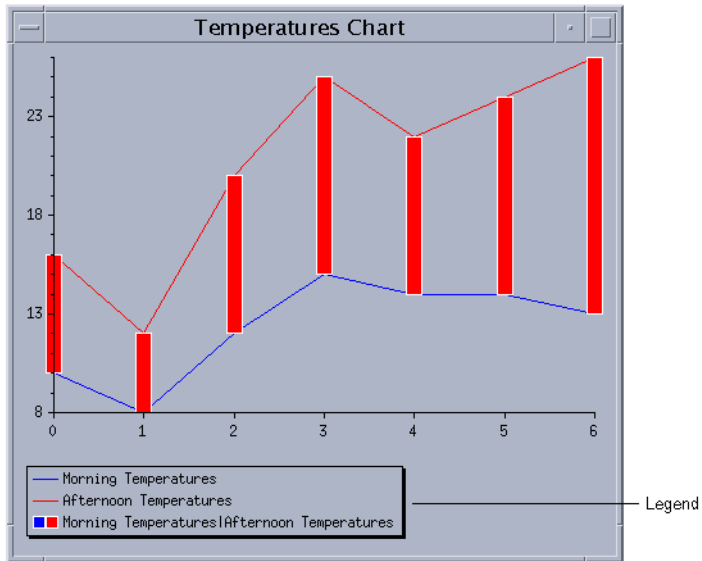


Figure 10.1 A Legend Added to a Chart

The legend items that make up the legend are represented by instances of the `IlvChartLegendItem` class. Each legend item is composed of two elements (see Figure 10.2):

- ◆ The *graphic part* illustrates the graphical representation of data with which the legend item is associated. For example, for data displayed by markers, the graphic part will show the marker with the same color and shape.
- ◆ The *text part* displays a description or a label for the represented data.

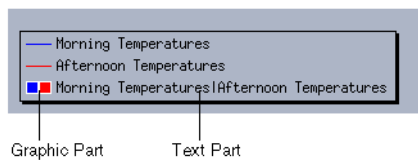


Figure 10.2 Elements of a Legend Item

Setting General Properties

The following table shows the properties defined for a chart legend object.

Property	Methods	Default Value
General Properties		
Surrounded by a Frame	isShowingFrame showFrame	IlvTrue
Drawn without a Background and a Shadow	isTransparent setTransparent	IlvFalse
Automatically Recomputed	isAutoRecomputing setAutoRecompute	IlvTrue
Automatically Fit to Contents	isAutoFitting setAutoFit	IlvTrue
Properties Related to the Legend Items		
Palette Used to Display the Legend Items	getItemPalette setItemPalette	0
Space Between the Area where the Graphic Part is Displayed and the Label of a Legend Item	getLabelSpacing setLabelSpacing	4
Space between the Legend Items	getItemSpacing setItemSpacing	4
Width of the Area where the Graphic Part of the Legend Items is Displayed	getBoxWidth setBoxWidth	20
Height of the Area where the Graphic Part of the Legend Items is Displayed	getBoxHeight setBoxHeight	10

By default, a legend is drawn surrounded by a filled rectangle with a shadow. This means that the flag indicating whether the legend is surrounded by a frame is set to `IlvTrue` and the flag indicating whether the legend is drawn without a background and a shadow is set to `IlvFalse`.

When a legend object is created, it does not contain any legend items. Since the property “Automatically Recomputed” is set to `IlvTrue` by default, the legend items corresponding to the displays defined in a chart will be computed automatically and added to the legend when the legend is set on the chart. Also, the legend items are rearranged when the legend is

resized, when the inner spacing of the legend is modified, and so on. New legend items are also added automatically and arranged when new displayers are added to the chart.

By default, the legend is resized automatically to fit the legend items that it contains because the property “Automatically Fit to Contents” is set to `IlvTrue`. If you do not want the size of the legend to be systematically recomputed from the legend items, you have to set the property “Automatically Fit to Contents” to `IlvFalse`.

Adding a Legend to a Chart

A chart legend object is added to a chart by means of the `IlvChartGraphic::setLegend` method. When the legend is set on the chart, the legend items corresponding to the displayers that are defined in the chart are computed automatically and added to the legend (if the property “Automatically Recomputed” has its default value `IlvTrue`) and the size of the legend is recomputed to fit these legend items (if the property “Automatically Fit to Contents” has its default value `IlvTrue`).

For an example of how to create and add a legend to a chart, see *Adding a Legend* on page 108 in the chapter “Chart Basics.”

***Note:** The chart legend object is a graphic object just like the chart object and is positioned independently of the chart object. Therefore, when you add the chart object to the container or manager that contains it, do not forget to add the chart legend object to this container or manager as well. Otherwise, the chart legend object will not appear on the screen.*

Displaying a Grid

A grid is a graphical indicator of data values. A grid is attached to a scale and is composed of the following:

- ◆ Major lines drawn at the positions of the major ticks of the scale
- ◆ Minor lines drawn at the positions of the minor ticks of the scale

A grid is displayed within a chart by a dedicated object called a *grid displayer*.

The base class used to represent a grid displayer is the `IlvAbstractGridDisplayer` class.

Figure 10.3 shows all the grid displayers defined in the Charts Library.

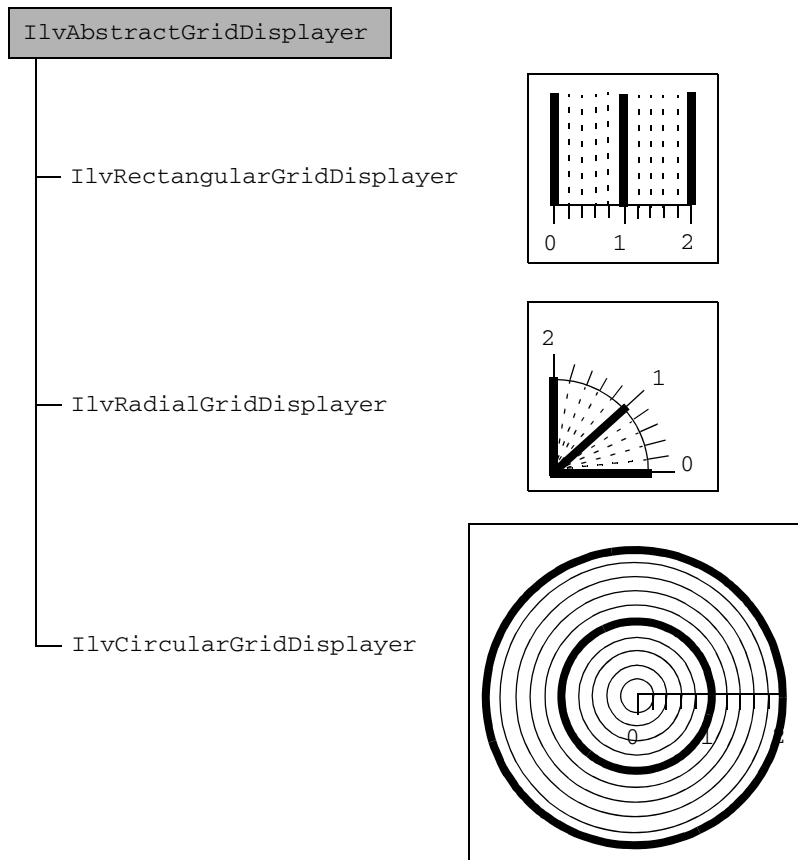


Figure 10.3 Hierarchy of Grid Displayers in the Charts Library

Setting General Properties

The following table shows the properties defined for all the grid displayers.

Property	Methods	Default Value
Visibility	isVisible setVisible	IlvTrue
Drawing Order Relative to the Drawing of the Graphical Representations of Data	getDrawOrder setDrawOrder	IlvDrawAbove

Property	Methods	Default Value
Palette Used to Draw the Major Lines	getMajorPalette setMajorPalette	0
Palette Used to Draw the Minor Lines	getMinorPalette setMinorPalette	0
Minor Lines Drawn	isDrawingMinorLines drawMinorLines	IlvFalse

A grid (like a scale) can be displayed on top of (IlvDrawAbove) or underneath (IlvDrawBelow) the graphical representations of data in a chart. By default, a grid will be displayed on top of the graphical representations. However, the drawing order can be changed for a given grid by means of the `IlvAbstractGridDisplayer::setDrawOrder` method.

If no palettes are defined to draw the major and/or the minor grid lines, the palette that will be used by default is the palette defined for the axis of the scale to which the grid is attached.

By default, a grid is drawn with only the major lines. To specify that the minor lines should also be drawn, you have to call the method `IlvAbstractGridDisplayer::drawMinorLines` with `IlvTrue` as a parameter.

Adding a Grid Displayer to a Scale

Since a grid is attached to a scale, the type of the grid used for a given scale depends on the type of the scale. The type of the scales that can be used for a given chart depends on the type of the projection used in the chart.

The following table lists the type of scale displayers and grid displayers that can be used for a Cartesian chart, which uses a Cartesian projection.

Chart type	Cartesian
Projection used	Cartesian -> IlvCartesianProjector

Scale Displayers	
Abscissa Coordinate	Rectangular -> IlvRectangularScaleDisplayer
Ordinate Coordinate(s)	Rectangular -> IlvRectangularScaleDisplayer
Grid Displayers	
Abscissa Coordinate	Rectangular -> IlvRectangularGridDisplayer
Ordinate Coordinate(s)	Rectangular -> IlvRectangularGridDisplayer

The following table lists the type of scale displayers and grid displayers that can be used for a polar chart, which uses a polar projection.

Chart type	Polar
Projection used	Polar -> IlvPolarProjector
Scale Displayers	
Abscissa Coordinate	Circular -> IlvCircularScaleDisplayer
Ordinate Coordinate(s)	Rectangular -> IlvRectangularScaleDisplayer
Grid Displayer(s)	
Abscissa Coordinate	Radial -> IlvRadialGridDisplayer
Ordinate Coordinate(s)	Circular -> IlvCircularGridDisplayer

Figure 10.4 shows examples of grids that are set on the scales of two charts. Grids are set on scales in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

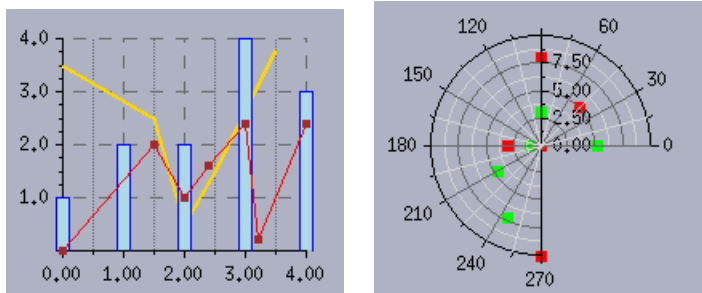


Figure 10.4 Grids in a Cartesian Chart and in a Polar Chart

To create and set a grid attached to a given scale, perform the following steps:

1. Create the grid displayer to display the grid that will be attached to the scale.

Do one of the following:

- Look at the tables at the beginning of this section to determine which grid can be used with the scale and create the corresponding grid displayer by hand.

or

- Use the `IlvSingleScaleDisplayer::createGridDisplayer` method of the scale displayer. This method will create the correct grid displayer to be used with the scale for you.

```
IlvAbstractGridDisplayer* gridDisplayer =
    scaleDisplayer->createGridDisplayer(referenceScaleDisplayer);
```

The scale displayer passed as a parameter is the displayer of the scale that must be used as a reference to know where the lines of the grid must stop. If the grid is attached to the scale representing the abscissa coordinate, the scale displayer that is passed as a parameter is the displayer of a scale representing an ordinate coordinate. If the grid is attached to a scale representing the ordinate coordinate, the scale displayer that is passed as a parameter is the displayer of the scale representing the abscissa coordinate.

2. Set the created grid displayer on the displayer of the scale to which you want to attach the grid.

```
scaleDisplayer->setGridDisplayer(gridDisplayer);
```

The Charts Library also provides some methods at the level of a chart object that directly encapsulate the creation and setting of a grid on a scale of the chart object. The following methods are available:

- ◆ `IlvChartGraphic::addAbscissaGrid` to add a grid on the abscissa scale.
- ◆ `IlvChartGraphic::addOrdinateGrid` to add a grid on an ordinate scale.

Displaying a Cursor

A cursor is a graphical indicator of a data value. A cursor is associated with a given scale and indicates the value corresponding to a given data point on this scale. It is composed of two distinct elements:

- ◆ A *delimiter* that crosses the data display area at the position of the data value
- ◆ An *axis mark* located on the scale. This axis mark indicates the data value at the level of the scale. It is made of a filled rectangle with a label.

The base class used to display a cursor is the `IlvAbstractChartCursor` class.

Figure 10.5 shows all the classes displaying cursors that are defined in the Charts Library.

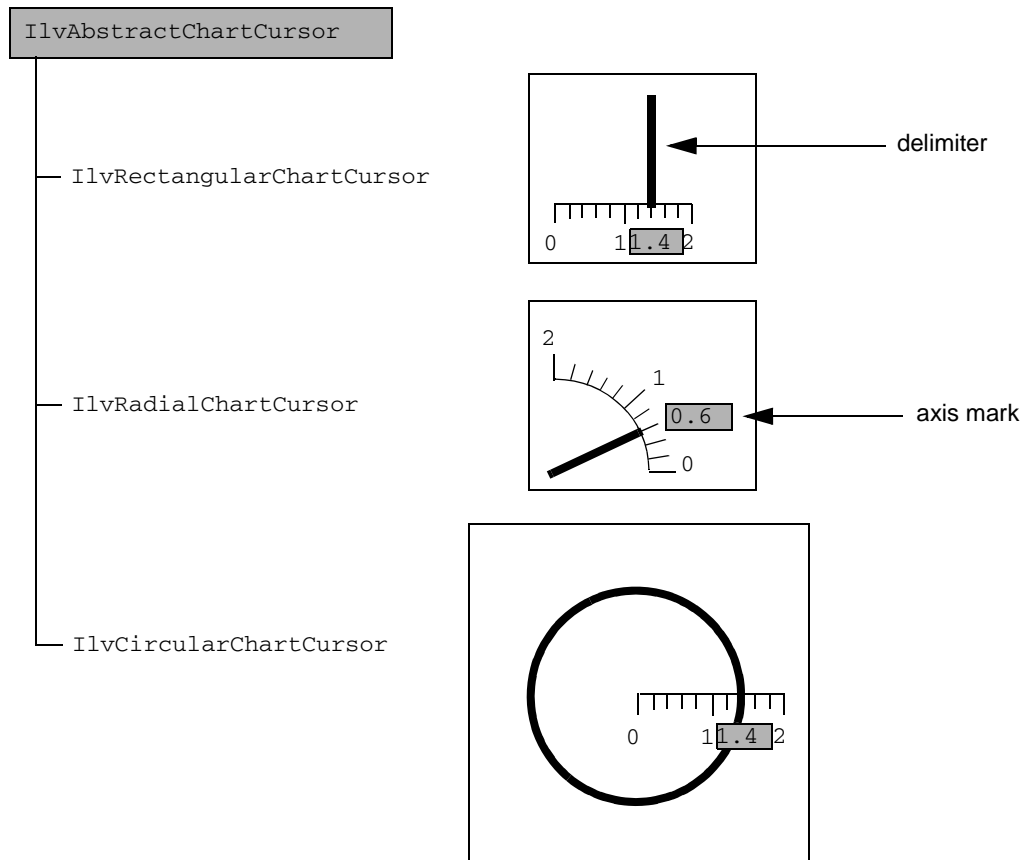


Figure 10.5 Hierarchy of Classes Displaying Cursors in the Charts Library

Setting General Properties

The following table shows the properties defined for all the classes displaying cursors.

Property	Methods	Default Value
Visibility	<code>isVisible</code> <code>setVisible</code>	<code>IlvTrue</code>
Drawing Order Relative to the Drawing of the Graphical Representations of Data	<code>getDrawOrder</code> <code>setDrawOrder</code>	<code>IlvDrawAbove</code>
Palette Used to Draw the Cursor	<code>getPalette</code> <code>setPalette</code>	<code>0</code>
Axis Mark Drawn	<code>isDrawingAxisMark</code> <code>drawAxisMark</code>	<code>IlvTrue</code>
Delimiter Drawn	<code>isDrawingDelimiter</code> <code>drawDelimiter</code>	<code>IlvTrue</code>
Delimiter Drawn in XOR Mode	<code>isDrawingGhost</code> <code>drawGhost</code>	<code>IlvFalse</code>
Data Value for which the Cursor is Drawn	<code>getValue</code> <code>setValue</code>	<code>0</code>
Label Displayed in the Axis Mark	<code>getLabel</code> <code>setLabel</code>	<code>0</code>

A cursor (like a grid or a scale) can be displayed on top of (`IlvDrawAbove`) or underneath (`IlvDrawBelow`) the graphical representations of data in a chart. By default, a cursor will be displayed on top of the graphical representations. However, the drawing order can be changed for a given cursor by means of the `IlvAbstractChartCursor::setDrawOrder` method.

If no palette is defined to draw the cursor, the palette that will be used by default is the one defined for the axis of the scale with which the cursor is associated.

The axis mark displays the defined cursor label, if it exists. This label is returned by the `IlvAbstractChartCursor::getLabel` method. However, the axis mark directly displays the data value for which the cursor is drawn. This data value is returned by the `IlvAbstractChartCursor::getValue` method.

To speed up the drawing process when the cursor is used as a dynamic mark (for example, when using it in a crosshair), you can specify that the delimiter is to be drawn in XOR mode by using the `IlvAbstractChartCursor::drawGhost` method.

Adding a Cursor to a Scale

You can set as many cursors as you want on a given scale. Since a cursor is associated with a scale, the type of the cursor used for a given scale depends on the type of the scale. The type of the scales that can be used for a given chart depends on the type of the projection used in the chart.

The following table lists the type of scale displayers and cursor displayers that can be used for a Cartesian chart, which uses a Cartesian projection.

Chart type	Cartesian
Projection used	Cartesian -> <code>IlvCartesianProjector</code>
Scale Displayers	
Abscissa Coordinates	Rectangular -> <code>IlvRectangularScaleDisplayer</code>
Ordinate Coordinate(s)	Rectangular -> <code>IlvRectangularScaleDisplayer</code>
Cursor Displayer(s)	
Abscissa Coordinates	Rectangular -> <code>IlvRectangularChartCursor</code>
Ordinate Coordinate(s)	Rectangular -> <code>IlvRectangularChartCursor</code>

The following table lists the type of scale displayers and cursor displayers that can be used for a polar chart, which uses a polar projection.

Chart type	Polar
Projection used	Polar -> <code>IlvPolarProjector</code>

Scale Displayers	
Abscissa Coordinate	Circular -> <code>IlvCircularScaleDisplayer</code>
Ordinate Coordinate(s)	Rectangular -> <code>IlvRectangularScaleDisplayer</code>
Cursor Displayers	
Abscissa Coordinate	Radial -> <code>IlvRadialChartCursor</code>
Ordinate Coordinate(s)	Circular -> <code>IlvCircularChartCursor</code>

Figure 10.6 shows examples of cursors that are set on the scales of two charts. Cursors are set on scales in a Cartesian chart (using a Cartesian projection) and in a polar chart (using a polar projection).

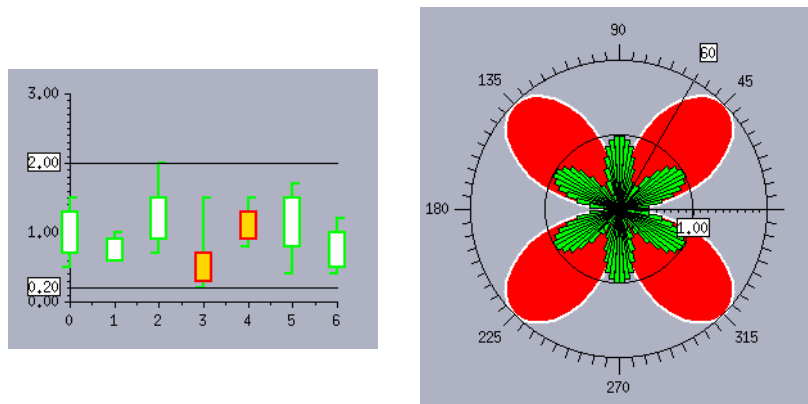


Figure 10.6 *Cursors in a Cartesian Chart and in a Polar Chart*

To create and set a cursor associated with a given scale, perform the following steps:

1. Create the object that will display the cursor associated with the scale.

Do one of the following:

- Look at the tables at the beginning of this section to determine which cursor can be used with the scale and create the corresponding cursor displayer by hand.

or

- Use the `IlvSingleScaleDisplayer::createCursor` method of the scale displayer. This method will create the correct cursor displayer to be used with the scale for you.

```
IlvAbstractChartCursor* cursorDisplayer =
    scaleDisplayer->createCursor(referenceScaleDisplayer);
```

The scale displayer passed as a parameter is the displayer of the scale that must be used as a reference to know where the delimiter of the cursor must stop. If the cursor is associated with the scale representing the abscissa coordinate, the scale displayer that is passed as a parameter is the displayer of a scale representing an ordinate coordinate. If the cursor is associated with a scale representing the ordinate coordinate, the scale displayer that is passed as a parameter is the displayer of the scale representing the abscissa coordinate.

2. Set the data value for which the cursor will be drawn.

```
cursorDisplayer->setValue(dataValue);
```

3. Set the object created to display a cursor on the displayer of the scale with which you want to associate the cursor.

```
scaleDisplayer->addCursor(cursorDisplayer);
```

The Charts Library also provides some methods at the level of a chart object that directly encapsulate the creation and setting of a cursor on a scale of the chart object. The following methods are available:

- ◆ `IlvChartGraphic::addAbscissaCursor` to add a cursor on the abscissa scale.
- ◆ `IlvChartGraphic::addOrdinateCursor` to add a cursor on an ordinate scale.

Interacting with Charts

The Charts Library provides interactors that allow the user to interact with a chart. This chapter provides detailed information on the chart interactors. You can find information on the following topics:

- ◆ *Using the Chart Interactors*
- ◆ *Setting an Interactor on a Chart Object*

Using the Chart Interactors

The base class used to define the behavior of a chart in response to a given action by the user is the `IlvChartInteractor` class.

Several subclasses are predefined in the Charts Library. Some of these subclasses inherit directly from the `IlvChartInteractor` class:

- ◆ `IlvChartZoomInteractor` allows the user to zoom in and zoom out on the data display area.
- ◆ `IlvChartScrollInteractor` allows the user to scroll the displayed data by using the arrow keys.
- ◆ `IlvChartPanInteractor` allows the user to scroll the displayed data by using the mouse.

- ◆ `IlvChartCrossHairInteractor` displays a crosshair at the location of the mouse pointer.

Other subclasses inherit from the `IlvChartDataInteractor` class, a subclass of `IlvChartInteractor` that specifically deals with interactions on the data points of the chart:

- ◆ `IlvChartDragPointInteractor` allows the user to drag a data point.
- ◆ `IlvChartHighlightPointInteractor` allows the user to trigger an action whenever the mouse moves over a data point in the data display area.
- ◆ `IlvChartInfoViewInteractor` inherits from the `IlvChartHighlightPointInteractor` class and displays information about a data point whenever the mouse moves over the data point in the data display area.
- ◆ `IlvChartSelectInteractor` allows the user to select data points (either all the data points of a data set or only a single data point) and then to trigger an action on the selected data point(s).

Note: The precision that is currently used to find the data point corresponding to a given screen point is returned by the `IlvChartDataInteractor::GetPrecision` method. This precision can be changed by means of the `IlvChartDataInteractor::SetPrecision` method.

Instances of the chart interactors can be either shared or individually instantiated. Shared instances are stored in a register and can be accessed by their registered name.

Note: Only shared chart interactor instances are persistent. The nonshared instances that you have instantiated individually are saved as if they were the shared instance of the same chart interactor class.

Predefined Chart Interactors

The following sections describe the chart interactors of the Charts Library. For each chart interactor, you will find a table that includes the registered name of the shared instance of the interactor, the key or button used for the interaction, and the action that is performed when using the interactor.

The following interactors are defined in the Charts Library:

- ◆ *Zoom Interactor*
- ◆ *Scroll Interactor*
- ◆ *Pan Interactor*
- ◆ *Crosshair Interactor*
- ◆ *Drag-Point Interactor*

- ◆ *Highlight-Data-Point Interactor*
- ◆ *Information-View Interactor*
- ◆ *Select-Data-Points Interactor*

Zoom Interactor

A zoom interactor has the following basic characteristics:

Class	<code>IlvChartZoomInteractor</code>
Registered name	“ChartZoom”
Key or Button	- Left mouse button to zoom in - Shift + Left mouse button to zoom out
Action	Lets the user trigger a zoom-in or a zoom-out command by dragging a box within the data display area of a chart. This box indicates the area to be zoomed in or zoomed out.

The mouse button used to perform the zoom-in and zoom-out operations is the left mouse button by default. However, this button can be changed by passing another button as a parameter to the constructor of the zoom interactor.

Each zoom-in/zoom-out operation can be broken down into several steps to render a smooth transition between the original and the final visual state of the displayed data. When a zoom interactor instance is created, the default number of steps is set to 0. You can specify the number of steps by means of the `IlvChartZoomInteractor::setZoomSteps` method.

Scroll Interactor

A scroll interactor has the following basic characteristics:

Class	<code>IlvChartScrollInteractor</code>
Registered name	“ChartScroll”
Key or Button	Arrow keys
Action	Lets the user scroll through the displayed data.

Pan Interactor

A pan interactor has the following basic characteristics:

Class	<code>IlvChartPanInteractor</code>
Registered name	"ChartPan"
Key or Button	Right mouse button
Action	Lets the user scroll through the displayed data by dragging the mouse in any direction.

The mouse button used to scroll through the displayed data is the right mouse button by default. However, this button can be changed by passing another button as a parameter to the constructor of the pan interactor.

Crosshair Interactor

A crosshair interactor has the following basic characteristics:

Class	<code>IlvChartCrossHairInteractor</code>
Registered name	"ChartCrossHair"
Key or Button	"C"
Action	Allows the user to trigger the visibility of a crosshair that will follow the movement of the mouse. The global visibility of the cursors that make up the crosshair is turned on or off when the "C" key is pressed.

Drag-Point Interactor

A drag-point interactor allows the user to drag a data point. It has the following basic characteristics:

Class	<code>IlvChartDragPointInteractor</code>
Registered name	"ChartDragPoint"
Inherits from	<code>IlvChartDataInteractor</code>
Key or Button	Left mouse button
Action	Lets the user modify a data point by dragging its graphical representation within the data display area.

The mouse button used to perform the drag operation is the left mouse button by default. However, this button can be changed by passing another button as a parameter to the constructor of the drag-point interactor.

You can use two modes for the drag operation:

- ◆ With the *opaque* mode, the data point is modified each time the mouse is dragged.
- ◆ With the *ghost* mode, the data point is modified only when the mouse button is released.

When a drag-point interactor is created, the default mode that is used for the drag operation is the ghost mode. You can specify that the opaque mode should be used by calling the method `IlvChartDragPointInteractor::setOpaque` with `IlvTrue` as a parameter.

Highlight-Data-Point Interactor

A highlight-data-point interactor allows the user to trigger an action whenever the mouse moves over a data point in the data display area. It has the following basic characteristics:

Class	<code>IlvChartHighlightPointInteractor</code>
Registered name	"ChartHighlightPoint"
Inherits from	<code>IlvChartDataInteractor</code>
Key or Button	None
Action	Allows the user to trigger an action whenever the mouse moves over a data point in the data display area.

No action is specified by default. The user has to specify by hand the action that should be triggered whenever the mouse moves over a data point in the data display area. This action is set by means of the `IlvChartHighlightPointInteractor::setAction` method. The action should be of the type `IlvChartHighlightPointInteractor::Action`:

```
typedef void (* Action )(IlvChartGraphic* chart,
                        IlvAbstractChartDisplayer* disp,
                        IlvChartDataSet* dataSet,
                        IlvUInt pointIndex,
                        IlvBoolean highlight);
```

Information-View Interactor

An information-view interactor displays information about a data point whenever the user moves the mouse over the data point. It has the following basic characteristics:

Class	<code>IlvChartInfoViewInteractor</code>
Registered name	"ChartInfoView"

Inherits from	<code>IlvChartHighlightPointInteractor</code>
Key or Button	None
Action	Displays information about a data point whenever the user moves the mouse over the data point in the data display area.

The information is displayed in a small window. The text that is displayed by default is the name of the data set to which the data point belongs and the abscissa and ordinate values of the data point. This text can be redefined in a subclass.

Select-Data-Points Interactor

A select-data-points interactor allows the user to select data points in the data display area. It has the following basic characteristics:

Class	<code>IlvChartSelectInteractor</code>
Registered name	"ChartSelect"
Inherits from	<code>IlvChartDataInteractor</code>
Key or Button	Left mouse button
Action	Allows the user to select data by clicking a projected point in the data display area and then to trigger an action on the selected data point(s).

The mouse button used to perform the selection is the left mouse button by default. However, this button can be changed by passing another button as a parameter to the constructor of the selection interactor.

When the user clicks a projected data point for the first time, all the data points of the data set to which the projected point belongs are selected. These data points are marked as selected with markers by default. When the user clicks one of the selected data points in this data set again, only the corresponding data point is then selected. This data point is marked as selected with a marker by default. The graphic objects drawn to mark data points as selected can be redefined in a subclass.

No action is specified by default. The user has to specify by hand the action to be triggered whenever data are selected or deselected. This action is set by means of the `IlvChartSelectInteractor::setAction` method. The action should be of the type `IlvChartSelectInteractor::Action`:

```
typedef void (* Action )(IlvChartGraphic* chart,
                        IlvAbstractChartDisplayer* disp,
                        IlvChartDataSet* dataSet,
```

```
IlvUInt pointIndex,
IlvBoolean select);
```

Setting an Interactor on a Chart Object

Several interactors can be used at the same time to interact with a given chart object. These interactors are managed by a dedicated object called a *chart interactor manager*.

The base class used to represent a chart interactor manager is the `IlvChartInteractorManager` class. Two methods are available to add an interactor to be managed by a chart interactor manager:

```
void addInteractor(IlvChartInteractor* interactor,
                  IlvUInt position=IlvLastPositionIndex)
```

and

```
void addInteractor(const char* name,
                  IlvUInt position=IlvLastPositionIndex)
```

The first method is used to add a nonshared interactor and the second method to add a shared interactor. The position at which the interactor is added can be specified. This allows you to indicate the priority for dispatching events since interactors are considered in the order of their indexes.

The basic steps to set interactors on a given chart object are the following:

1. Create a chart interactor manager to handle the interactors that will be used to interact with the chart object.
2. Add the interactors to be used with the chart to the created chart interactor manager.
3. Attach the chart interactor manager to the chart object.

Example

The `stock` sample demonstrates the use of interactors. The source code of this sample can be found in the `stock.cpp` file in the `$ILVHOME/samples/charts/interactors/src` directory. This section describes only the steps required to set interactors on the chart.

Creating a Chart Interactor Manager to Manage the Interactors

To create a chart interactor manager to manage the interactors, we use the following code:

```
IlvChartInteractorManager* interMgr = new IlvChartInteractorManager();
```

Adding Interactors to the Chart Interactor Manager

For this example, we are going to add a zoom interactor, a pan interactor, and a scroll interactor to the chart interactor manager.

1. First, we add the zoom interactor. We are going to create our own instance of the interactor because we want to set a particular number of intermediate steps for the zoom-in/zoom-out operation.

```
IlvChartZoomInteractor* zinter = new IlvChartZoomInteractor(); // not
shared
zinter->setZoomSteps(4);
interMgr->addInteractor(zinter);
```

2. Next, we add the pan interactor. We are going to use the shared instance because the default settings of this instance correspond to what we want.

```
interMgr->addInteractor("ChartPan");
```

3. Finally, we add the scroll interactor. We also use the shared instance.

```
interMgr->addInteractor("ChartScroll");
```

Attaching the Chart Interactor Manager to the Chart Object

To attach the chart interactor manager to the chart object, we use the following code:

```
IlvChartInteractorManager::Set(chart, interMgr);
```

Using Charts to Display Real-Time Data

The Charts Library provides the capability to display real-time data in your charts. This chapter gives detailed information on how to use the scroll modes to do this.

You will find information on the following topics:

- ◆ *Automatic Scroll Modes*
- ◆ *Using Automatic Scroll Modes to Display Real-Time Data*
- ◆ *Improving Performance When Adding Data Points to a Chart*

Automatic Scroll Modes

A chart object can display either predefined data or real-time data (data arriving “on the fly”). The way a given chart object reacts when new data are added is defined by a value of type `IlvChartGraphic::ScrollMode`. Three scroll modes have been predefined:

- ◆ `IlvScrollModeStop`

When new data items are added, the chart does not scroll. The new data items that are added are displayed only if they belong to the data display area.

- ◆ `IlvScrollModeShift`

When a new data item is added, the chart scrolls if the data item is out of the displayed range of the abscissa coordinate. The scrolling operation is performed along the abscissa scale in the direction of the decreasing values. When the chart scrolls, it shifts by the number of scroll ratios necessary to display the new data item in the data display area. (Figure 12.1 and Figure 12.2 show examples of a chart that uses shift scroll mode.)

◆ `IlvScrollModeCycle`

When a new data item is added, the chart scrolls if the data item is out of the displayed range of the abscissa coordinate. In the case of cycle mode, when a new data item is added that is out of the displayed range of the abscissa coordinate, this data item is simply displayed in the area where the minimum values are displayed, thus erasing these values. So the data is displayed cyclically as it arrives. When the chart scrolls, it scrolls by the number of scroll ratios necessary to display the data item in the data display area.

When you create a chart object, the scroll mode that is set by default is `IlvScrollModeStop`. You can change the scroll mode with the `IlvChartGraphic::setScrollMode` method. If the new scroll mode is set to `IlvScrollModeCycle`, you can also specify that a scrolling cursor that marks the beginning of the cycle should be displayed as well. Do this by setting the parameter `createCursor` of the `IlvChartGraphic::setScrollMode` method to `IlvTrue`.

When you create a chart object, the scroll ratio is set to 0.25 by default. This means that the portion of the chart that is scrolled corresponds to a quarter of the range of the abscissa coordinate. You can change the scroll ratio by means of the `IlvChartGraphic::setScrollRatio` method. The scroll ratio should be between 0 and 1.

Using Automatic Scroll Modes to Display Real-Time Data

To use automatic scroll modes to display data arriving on the fly (real-time data), do the following:

1. Define the range of the abscissa coordinate that will be displayed.
This is required because the scrolling operation is performed along the abscissa coordinate.
2. Define the range of the ordinate coordinate that will be displayed.
Although this step is not required, it is useful because it prevents the range of values represented by the ordinate scale from being constantly updated to fit the displayed data.
3. Set the scroll mode (shift mode or cycle mode) to be used.
4. Set the scroll ratio to be used.

The next section contains an example that illustrates these basic steps.

Scroll Example

The following example shows you how to display three sets of data arriving on the fly. The data sets are empty when created and are set on three polyline displayers. A data point is added to each data set at each period of a timer. The shift scroll mode is used in this example.

The following images show the example chart at two different periods of the timer. Figure 12.1 shows the chart before the scrolling operation. Figure 12.2 shows the chart after it has been scrolled. The chart shifts according to the value defined by the scroll ratio.



Figure 12.1 Chart before the Scrolling Operation in Shift Scroll Mode

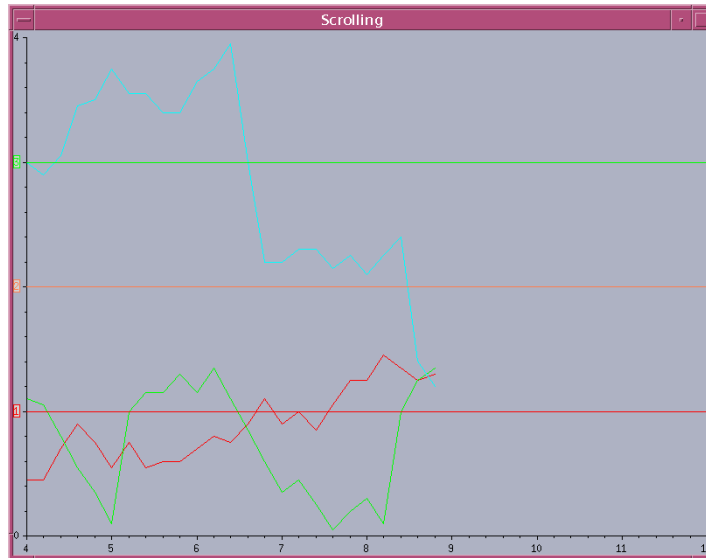


Figure 12.2 *Chart after Scrolling by the Scroll Ratio in Shift Scroll Mode*

The complete source code for this example can be found in the `scroll11.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory. This section describes only the steps required to use the scroll modes to display data coming on the fly.

Defining the Range of the Abscissa Coordinate

The range of values that will be displayed for a given coordinate is set on the coordinate information object associated with this coordinate. To define the range of the abscissa coordinate, we can use the following code:

```
chart->getAbscissaInfo()->setUserDataRange( IlvCoordInterval(0,8) );
```

This specified interval is the one that is displayed for the abscissa coordinate at the beginning of the data display (see Figure 12.1). This interval will change when the chart scrolls. However, the difference between the maximum and minimum values of the displayed interval will remain the same (see Figure 12.2).

Defining the Range of the Ordinate Coordinate

To define the range of the ordinate coordinate, we can use the following code:

```
chart->getOrdinateInfo()->setUserDataRange( IlvCoordInterval( MinOrd, MaxOrd) );
```

Setting the Scroll Mode

For this example, we want to use the shift scroll mode. To set the scroll mode, we can use the following code:

```
chart->setScrollMode(IlvChartGraphic::IlvScrollModeShift);
```

Setting the Scroll Ratio

We want to set the scroll ratio to 0.5. This means that half of the displayed range of the abscissa coordinate will be shifted when necessary. To set the scroll ratio, we can use the following code:

```
chart->setScrollRatio(.5);
```

Adding the Data on the Fly

Data is added on the fly by using a method that is called at each period of a timer. This method simply adds a new data point to each defined data set. The chart will be automatically updated to reflect the addition of data points without doing anything more.

For more information on how data is automatically updated, see *Modifying Data and Updating Charts* on page 134.

The method that is called at each period of the timer is the following:

```
static void
AddPoints(IlvTimer* timer, IlAny arg)
    // Timer callback to add points.
{
    IlvChartGraphic* chart = (IlvChartGraphic*)arg;
    IlvDisplay* display = timer->getDisplay();
    IlUInt count;
    IlvDoublePoint previousPoint;
    IlvDoublePoint nextPoint;
    IlvChartDataSet* dataSet;
    // Add one point on each dataSet.
    for (IlUInt i = 0; i < chart->getDataSetsCount(); i++) {
        dataSet = chart->getDataSet(i);
        count = dataSet->getDataCount();
        // Get the previous data point to have a smooth random creation.
        if (count != 0) {
            dataSet->getPoint(count-1, previousPoint);

            // Create a new data point.
            GeneratePoint(previousPoint, nextPoint);
        }
        else {
            nextPoint.x(0);
            nextPoint.y(i+0.5);
        }

        // Add the data point to the data set.
        dataSet->addPoint(nextPoint);
    }
}
```

Improving Performance When Adding Data Points to a Chart

By default, each time a data point is added to a data set, the chart is updated immediately. The area of the chart where the graphical representation of the new data point should be displayed is first invalidated and then redrawn by the holder of the chart. The holder is an instance of the `IlvGraphicHolder` class and can be retrieved for a given graphic object by the `getHolder` method.

If you look at our example, the redrawing of the chart that is required by adding the data points will be performed three times (one time for each data set that is modified) since we add a data point to three data sets at each period of the timer.

There are several methods that you can use to improve performance when adding data points to a chart:

- ◆ Use the `initReDraws` and `reDrawViews` methods of the holder of the chart.
This allows you to perform the redraw operation for all the data sets at one time instead of performing the redraw operation for each data set to which a data point is added.
- ◆ Use the fast scroll mode of the chart.
This allows you to draw the graphical representation of the newly added data point directly in the drawing port without going through the holder of the chart.
- ◆ Batch the modifications when adding new data points to a data set.
- ◆ Use an `IlvChartCyclicPointSet` data set.

Note: Batching modifications can be used alone or can be used in conjunction with the use of the fast scroll mode.

Using `initReDraws` and `reDrawViews` Methods

The `initReDraws` and `reDrawViews` methods allow you to specify that the redraw operation required by adding data points should be performed only one time. To use these methods, you need to do the following:

1. Add a call to the `initReDraws` method of the chart holder before the loop that adds a data point to each data set.
2. Add a call to the `reDrawViews` method of the chart holder after the loop that adds a data point to each data set.

In our scrolling example, each time a data point is added to a data set, the area of the chart that must be redrawn is simply invalidated and the redrawing required for the newly added data points for a given period of the timer will be performed one time when the `reDrawViews` method is called.

In our example, the method that is called at each period of the timer is now:

```
static void
AddPoints(IlvTimer* timer, IlvAny arg)
    // Timer callback to add points.
{
    IlvChartGraphic* chart = (IlvChartGraphic*)arg;
    chart->getHolder()->initReDraws();

    IlvDisplay* display = timer->getDisplay();
    IlvUInt count;
    IlvDoublePoint previousPoint;
    IlvDoublePoint nextPoint;
    IlvChartDataSet* dataSet;
    // Add one point on each dataSet.
    for (IlvUInt i = 0; i < chart->getDataSetsCount(); i++) {
        dataSet = chart->getDataSet(i);
        count = dataSet->getDataCount();
        // Get the previous data point to have a smooth random creation.
        if (count != 0) {
            dataSet->getPoint(count-1, previousPoint);

            // Create a new data point.
            GeneratePoint(previousPoint, nextPoint);
        }
        else {
            nextPoint.x(0);
            nextPoint.y(i+0.5);
        }

        // Add the data point to the data set.
        dataSet->addPoint(nextPoint);
    }

    chart->getHolder()->reDrawViews();
}
```

The complete source code can be found in the `scroll12.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory.

Using the Fast Scroll Mode

Using the fast scroll mode is another way to improve performance. This is done by calling the method `IlvChartGraphic::enableFastScroll` with `IlvTrue` as a parameter. Each time a data point is added to a data set, the graphical representation of the newly added data point is drawn directly in the drawing port without going through the holder of the chart.

Although fast scroll mode speeds the drawing operations, it has several limitations:

- ◆ There must be no other graphic objects underneath the chart object.

- ◆ Using displayers that display filled polygonal representations, such as polygon and stair displayers, should be avoided since an additional vertical line is drawn for each polygonal representation of an added data point.

Note: *The fast scroll mode can be used only with the `IlvScrollModeShift` scroll mode.*

Batching Modifications

You can drastically improve performance by using a batch operation when adding new data points to a data set. This is especially useful when it is not necessary to see the updates resulting from adding the new data points one by one as they occur. To batch the modifications, do the following:

1. Add a call to the `IlvChartDataSet::startBatch` method on the data set before a set of additions of data points is performed on the data set.

Once the `IlvChartDataSet::startBatch` method has been called, the updates resulting from adding new data points are no longer performed.

2. Add a call to the `IlvChartDataSet::endBatch` method on the data set after a set of additions of data points has been performed on the data set.

When the `IlvChartDataSet::endBatch` method is called, all the data points that have been added since the call to the `IlvChartDataSet::startBatch` method are processed at one time. The update resulting from the addition of all these data points is performed at one time. The newly added data points are drawn all at one time, instead of being drawn one by one as is the case when the modifications are not batched.

In our example, the method that is called at each period of the timer is now:

```
static void
AddPoints(IlvTimer* timer, IlvAny arg)
    // Timer callback to add points.
{
    IlvChartGraphic* chart = (IlvChartGraphic*)arg;

    IlvDisplay* display = timer->getDisplay();
    IlvUInt count;
    IlvDoublePoint previousPoint;
    IlvDoublePoint nextPoint;
    IlvChartDataSet* dataSet;
    // Add one point on each dataSet.
    for (IlvUInt i = 0; i < chart->getDataSetsCount(); i++) {
        dataSet = chart->getDataSet(i);

        dataSet->startBatch();

        count = dataSet->getDataCount();
        for (IlvUInt k = 0; k < NbAddedPts; k++, count++) {
            // Get the previous data point to have a smooth random creation.
            if (count != 0) {
                dataSet->getPoint(count-1, previousPoint);

                // Create a new data point.
                GeneratePoint(previousPoint, nextPoint);
            }
            else {
                nextPoint.x(0);
                nextPoint.y(i+0.5);
            }

            // Add the data point to the data set.
            dataSet->addPoint(nextPoint);
        }

        dataSet->endBatch();
    }
}
```

The complete source code can be found in the `scroll13.cpp` file located in the `$ILVHOME/samples/charts/userman/src` directory. In this source code, we use both the fast scroll mode and the batching modifications method when adding new data points.

Releasing the Automatic Update

The automatic update of the charts that is performed when data are modified can be released by calling the method `IlvChartGraphic::reDrawWhenNotified` with `IlvFalse` as a parameter. In this a case, the redrawing will have to be performed manually by calling the `IlvChartGraphic::updateAndReDraw` method.

A***The IlvXMLChartData Class***

In this Appendix you will find a detailed description of the `IlvXMLChartData` class.

Introducing the IlvXMLChartData Class

The class `IlvXMLChartData` derives from `IlvAbstractChartData` and reads one or several data sets from an XML file. The XML file should conform to the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT chartData (data+)>
<!ATTLIST chartData xmlns:ilvchart CDATA #FIXED
"http://www.ilog.com/products/jviews/chart"
        version CDATA #REQUIRED>

<!ELEMENT data (series+)>
<!ATTLIST data          xSeries IDREF #IMPLIED>

<!ELEMENT series ((value | valuesList)*, property*)>
<!ATTLIST series dateFormat CDATA          #IMPLIED
                    type (double | date)    #REQUIRED
                    id          ID          #REQUIRED>

<!ELEMENT value (#PCDATA)>

<!ELEMENT valuesList (#PCDATA)>
<!ATTLIST valuesList delimiter CDATA #IMPLIED>
<!ENTITY % propertyExt "">
<!ELEMENT property (#PCDATA %propertyExt;)*>
<!ATTLIST property name CDATA #REQUIRED
                    value CDATA #IMPLIED>

<!ELEMENT seriesRef EMPTY>
<!ATTLIST seriesRef ref IDREF #REQUIRED>
```

For example, assuming that a chart contains three data sets: two (DS_A and DS_B) which do not use `xvalues` series, and one (DS_C) which uses a specific series for its abscissa, the resulting XML file is:

```
<pre>
<?xml version="1.0" encoding="UTF-8"?>
<chartData version="0.3">
  <data>
    <series id="DS_A" type="double">
      <valuesList>0.0,8.0,6.0,13.0,22.0,21.0,19.0,28.0,27.0,23.0</valuesList>
    </series>
    <series id="DS_B" type="double">
      <valuesList>0.0,9.0,11.0,14.0,11.0,16.0,19.0,21.0,12.0,12.0</valuesList>
    </series>
  </data>
  <data xSeries="X_DS_C">
    <series id="X_DS_C" type="double">
      <valuesList>0.0,2.0,4.0,6.0,8.0,10.0,12.0,14.0,16.0,18.0</valuesList>
    </series>
    <series id="DS_C" type="double">
      <valuesList>0.0,0.0,6.0,3.0,0.0,2.0,9.0,18.0,9.0,5.0</valuesList>
    </series>
  </data>
</chartData>
</pre>
```

Tags Definition

In the next paragraphs, you will find the definition of the following tags:

- ◆ data
- ◆ series
- ◆ valuesList
- ◆ valueOperator
- ◆ property

data

The `<data>` tag defines a data set, that is, a set of series. It is possible to define one of the series as being the abscissa values, using the `xSeries` attribute:

```
<data xSeries="X_DS_C">
  ...
  <series id="X_DS_C">
    <valuesList>...</valuesList>
  </series>
</data>
```

series

The `<series>...</series>` tag has the following attributes:

- ◆ `id`: the identifier of the series, can be any string formed with alphanumeric characters and the underscore ('_').
- ◆ `type`: can be `double` or `date`.

If `double`, the values will be read as floating point numbers, using the dot ('.') as decimal separator.

If `date`, the values will be read as calendar dates, in `dd/mm/yy` format (31/12/2001). It is possible to use other formats if needed (see the `IlvXMLValueListProcessor` class).

valuesList

The `<valuesList>...</valuesList>` tag can hold either floating point values, or calendar dates (in `dd/mm/yy` format). It accepts only the attribute `delimiter` which defines the value delimiter in the list. The default is ','.

For example:

```
<valuesList>0.0,9.0,11.0,14.0,11.0,16.0,19.0,21.0,12.0,12.0</valuesList>
<valuesList delimiter="|">0.0|0.0|6.0|3.0|0.0|2.0|9.0|18.0|9.0|5.0</valuesList>
```

valueOperator

The `<valueOperator>` tag is only used to define a list of properties which can be processed separately through custom classes (see the `IlvXMLPropertyReader` class) and will be applied to the series for which the properties are defined.

For example:

```
<valueOperator>
  <property name="period">5</property>
</valueOperator>
```

property

Example:

```
<series id="X_DS_C">
  <valuesList>...</valuesList>
  <property name="period">5</property>
</series>
```

The `<property>` tag is used to declare properties which are processed by a property reader object (see the `IlvXMLPropertyReader` class). The object should be registered to the

`IlvXMLChartData` object through the `registerPropertyReader()` method prior for the specified property name to parsing the XML file (see `IlvXMLChartData::parse()`).

When the `IlvXMLChartData` object encounters a property, it checks if a property reader object was registered for the specified property name. If this is the case, it will call the reader `readProperty()` virtual method and store the returned `IlvXMLProperty` object. Once the parsing is completed, the `IlvXMLChartData` object will invoke the reader `setProperty()` virtual method, passing it the `IlvChartDataSet` which is being built.

It is up to the user to create his own property reader classes by deriving from `IlvXMLPropertyReader` and overriding the `readProperty()` and `setProperty()` methods according to his needs.

◆ `readProperty()`

turns an `IlxmlElement` into an `IlvXMLProperty`

◆ `setProperty()`

performs an operation on an `IlvChartDataSet` being given a property name and its value.

Note that the `IlvXMLProperty` class is also extensible.

Customizing Value and Date List Processing

If you need a specific treatment of any value lists, it is possible to give your own value list processor to the `IlvXMLChartData` class. To do this, derive `IlvXMLValueListProcessor` and override the `processValueList()` method. Then use `IlvXMLChartData::registerValueListProcessor()` to get it used by the `IlvXMLChartData` object.

Note: You can specify any kind of type name to be processed. The ones that are already defined are `double` or `date`, but you can create your own type if you need to.

For example:

```
<series id="X_DS_C" type="my_type">
  <valuesList>a,d,e,f,i,l</valuesList>
</series>
```

would be parsed with a `MyValueListProcessor` object, which would be registered as follows:

```
xmlChartData->registerValueListProcessor(IlString("my_type"),
                                         new MyValueListProcessor);
```

This is particularly useful if you need to parse date values written in a different format than the default one (`dd/mm/yy`). This is slightly easier using the `IlvXMLDateListProcessor`

A. The IlvXMLChartData Class

class, which has a specific method to override,
`IlvXMLDateListProcessor::parseDate()`.

So, supposing you need to parse:

```
<series id="X_DS_C" type="date">  
  <valuesList>01dec90,12jan91,14jul91,25sep91,15aug92</valuesList>  
</series>
```

You would then have your own `MyDateListProcessor` class with its own `parseDate()` method, and register it by using:

```
xmlChartData->registerValueListProcessor(IlString("my_type"),  
                                         new MyDateListProcessor);
```

It will then replace the default date list processor and be used instead.

Some examples of `IlvXMLChartData` can be found in the `samples/xml` directory.

Index

Numerics

3D bar displayer **163**

A

abscissa scale, customizing

Cartesian charts **44, 67**

polar charts **85**

abscissa scales

customizing **106, 111**

abscissa values

setting to radians **74**

Add icon **22**

addAbscissaCursor method

IlvChartGraphic class **228**

addAbscissaGrid method

IlvChartGraphic class **223**

addDataSet method

IlvAbstractChartData class **135**

adding

cursors to scales **226**

data on the fly **242**

displayers to charts **183**

graphic information to a data point **185**

grid displayers to scales **221**

interactors to interactor managers **236**

real-time data **242**

scale displayers to a chart **208**

addOrdinateCursor method

IlvChartGraphic class **228**

addOrdinateGrid method

IlvChartGraphic class **223**

addPoint method

IlvChartDataSet class **134**

analytic functions, charting **60**

annotations

descriptions of **97**

applying

transformations **213**

areListenersEnabled method

IlvChartDataSet class **137, 139**

automatic scroll modes **91**

automatic updates released **246**

Axis display, Scales subpage **51, 55**

definition **29**

B

Background window **58**

bar displayer **162**

batch operations for adding data items **138**

batch operations for modifying data **245**

C

callback function

creating a data set object **129**

Callbacks, Chart Inspector page **34**

Callbacks, Legend inspector page **35**

- Cartesian charts
 - adding legends to **108**
 - creating **40, 103, 105**
 - customizing
 - abscissa scale **44**
 - ordinate scale **46, 49, 52**
 - defining
 - data sets **41**
 - displayers **43**
 - definition of **91, 96**
 - example of chart layout **145**
 - representing analytic functions **60**
 - scales orientation **210**
 - simple example **103**
 - Temperatures Chart example **103**
 - types of grid displayers **221**
 - types of scale displayers **221**
- changing
 - orientation of scales **210**
- chart classes
 - Cartesian **101**
 - description of **101**
 - polar chart **102**
- chart data objects
 - adding data sets to **105, 111, 131**
 - basic rules for **99**
 - default instantiation **117**
 - description of **99, 115, 117**
 - user-defined **132**
- Chart inspector
 - icons **22**
 - overview **20**
- chart interactor managers
 - adding interactors **236**
 - attaching to chart objects **237**
- chart interactors
 - description of **230**
 - list of **230**
- chart layout
 - Cartesian chart example **145**
 - computing **144**
 - data display area defined **144**
 - drawing area defined **144**
 - general properties **146**
 - getting the chart layout object **148**
 - graph area defined **145**
 - polar chart example **145**
 - setting the chart layout object **148**
- chart objects
 - attaching chart interactor managers **237**
 - basic rules for **99**
 - components of **115**
 - description of **99, 116**
 - diagram of components **116**
 - setting interactors **236**
 - using with component classes **120**
- chart updates
 - as handled by chart data objects **135**
 - description of update process **137**
- charts
 - adding a legend **108, 112**
 - adding legends **219**
 - adding scale displayers **208**
 - additional decorations **97**
 - applying transformations to data values **213**
 - basic steps for creating **102**
 - creating a Cartesian **103**
 - creating a polar **109**
 - creating from a chart graphic object **120**
 - customizing **112**
 - definition of **96**
 - description of listener mechanism **137**
 - displaying legends **216**
 - displaying real-time data **239**
 - function in IBM ILOG Views **115**
 - how updates are performed **137**
 - listener mechanism **135**
 - setting interactors **236**
 - updating **135**
- Charts Library
 - component classes **117**
 - data features **91**
 - data modifications **134**
 - decoration features **92**
 - example images of charts **94**
 - features of the scales **92**
 - general architecture of **99**
 - global chart characteristics **90, 92**
 - handling data storage **126**
 - interactor features **93**

- list of graphic representations of data **91**
- list of interactors **230**
- model-view separation concept **99**
- sharing data **132**
- updating charts **135**
- Charts palette **18**
- circular displayer **205**
- Clean icon **22**
- component classes
 - in `IlvChartGraphic` objects **120**
- coordinate information objects
 - creating for abscissa **121**
 - creating for ordinate **122**
 - default instantiation **119**
 - description of **115, 119**
- creating
 - chart interactor managers **236**
 - cursors **227**
 - grids **223**
 - pie chart example **169**
 - point information object **187**
- creating a chart **20**
- cursor displayers
 - general properties **225**
 - hierarchy of classes **224**
 - types for Cartesian charts **226**
 - types for polar charts **226**
- cursors
 - adding to a scale **226**
 - creating and setting **227**
 - description of **97, 224**
 - displaying **224**
 - example of **97**
- customizing
 - scales **210**

D

- data classes
 - description of **100**
- data display
 - adding graphic information to a data point **185**
 - customizing **185**
 - drawing graphical representations **150**
- data display area **144**

- definition of **124**
- data modifications
 - as handled by chart data objects **134**
 - at chart data object level **134**
 - at data set level **134**
 - description of **134**
 - description of propagation process **138**
 - using batch operations **138**
 - using listeners to catch data **138**
- data point
 - adding graphic information **185**
 - adding point information objects **188**
 - out-of-bounds **157**
- data points
 - diagram of mapping into screen coordinates **124**
 - improving performance when adding to charts **243**
 - projected into screen coordinates **124**
 - projecting out-of-bounds points **191**
 - transformed into screen coordinates **123**
- data sets
 - adding a data set listener object **142**
 - adding to a chart data object **131**
 - adding to chart data objects **105, 111**
 - adding to default chart data object **131**
 - adding to user-defined chart data objects **132**
 - creating **104, 110**
 - definition of real **153**
 - definition of virtual **153**
 - function **101, 128**
 - putting data into **104**
 - set-of-points **101, 127**
 - set-of-values **101, 127**
 - types of **100, 127**
 - using a callback function **129**
 - using a script function **129**
- Data sets, Chart inspector page **23, 41**
- data sets, defining
 - by a script function **60**
 - Cartesian charts **41, 62**
 - polar charts **82**
- data sharing
 - as handled by chart data objects **132**
 - description of **132**
 - lock/unlock system **133**
- data storage

- as handled by chart data objects **126**
- description of **126**
- data values
 - applying transformations **213**
- dataPointAdded method
 - IlvChartDataListener class **139**
 - IlvChartDataSetListener class **139**
- dataPointChanged method
 - IlvChartDataListener class **139**
 - IlvChartDataSetListener class **139**
- dataPointRemoved method
 - IlvChartDataListener class **139**
 - IlvChartDataSetListener class **139**
- dataSetAdded method
 - IlvChartDataListener class **139**
- dataSetChanged method
 - IlvChartDataListener class **139**
- dataSetRemoved method
 - IlvChartDataListener class **139**
- defining
 - drawing order of scales **197**
 - minimum and maximum values for scales **212**
 - palettes **190**
 - position of labels **203**
 - scale steps and substeps **200**
 - tick position **202**
- defining step labels **201**
- description of **243**
- displayer factory **172**
 - code example **180**
- displayer model **171**
 - code example **180**
- displayers
 - adding to chart objects **105, 111**
 - adding to charts **183**
 - and ordinate scale **51**
 - composite **151**
 - description of **171**
 - high-low open-close **175**
 - marked polyline **173**
 - side-by-side **181**
 - stacked **177**
 - stacked 3D bar chart **177**
 - stacked bar chart **177**
 - stacked polygon **177**

- creating **105, 111**
- customizing **184**
- default instantiation **117**
- defining
 - Cartesian charts **43, 65**
 - polar charts **84**
- description of **99, 115, 117, 150**
- general properties **153**
- hierarchy diagram **151**
- polyline **105**
- setting colors **106**
- single **150**
 - 3D bar **163**
 - bar **162**
 - general properties **154**
 - high-low **164**
 - high-low bar **166**
 - pie **167**
 - polygon **159**
 - polyline **156**
 - scatter **155**
 - stair **161**
 - step **160**
- transforming data points **123**
- Displayers, Chart inspector page **24, 43, 57, 65**
- display-information interactor **234**
- displaying
 - grids **219**
 - legends **216**
 - real-time data **239**
- drag-point interactor **233**
- drawGhost method
 - IlvAbstractChartCursor class **225**
- drawing
 - overlapping step labels **203**
 - scales **194**
- drawing area **144**
- drawLabelOnCrossings method
 - IlvAbstractScaleDisplayer class **203**
- drawMinorLines method
 - IlvAbstractGridDisplayer class **221**
- drawOverlappingLabels method
 - IlvAbstractScaleDisplayer class **203**

E

enableFastScroll method
 IlvChartGraphic class **244**
enableListeners method
 IlvChartDataSet class **137**
endBatch method
 IlvChartDataSet class **138, 245**

F

fast scroll mode **243**
 using to improve performance **244**
function data set
 description of **128**
functions
 evaluation **63**

G

General, Chart inspector page **21**
General, Legend inspector page **35**
General, Scales subpage
 definition **28**
getAbscissaInfo method
 IlvChartGraphic class **212**
getCoordinateInfo method
 IlvAbstractScaleDisplay class **212**
getData method
 IlvChartGraphic class **105, 131**
getDisplay method
 IlvCompositeChartDisplay class **172**
getHiLoDisplay method
 IlvHiLoOpenCloseChartDisplay class **175**
getHolder method
 IlvGraphic class **243**
getLabel method
 IlvAbstractChartCursor class **225**
getLayout method
 IlvChartLayout class **148**
getLineDisplay method
 IlvMarkedPolylineDisplay class **173**
getMarkerDisplay method
 IlvMarkedPolylineDisplay class **173**
getOpenCloseDisplay method

 IlvHiLoOpenCloseChartDisplay class **175**
getOrdinateInfo method
 IlvChartGraphic class **212**
getOrientedClockwise method
 IlvPolarProjector class **212**
getParentDisplay method
 IlvAbstractChartDisplay class **154**
getProjectedPointsPalette method
 IlvPolylineChartDisplay class **157**
getStepLabelFormat method
 IlvSingleScaleDisplay class **201**
getUserDataMax method
 IlvCoordinateInfo class **213**
getUserDataMin method
 IlvCoordinateInfo class **213**
getValue method
 IlvAbstractChartCursor class **225**
graph area **145**
grid displays **219**
 adding to a scale **221**
 general properties of **220**
 hierarchy of classes **219**
 types for Cartesian charts **221**
 types for polar charts **222**
Grid, Scales subpage **67, 68**
 definition **29**
grids
 components of **219**
 creating and setting to a scale **223**
 description of **97**
 displaying **219**

H

high-low bar display **166**
high-low display **164**
high-low open-close display **175**

I

IBM ILOG Views Studio
 creating charts with **91**
icons, in the Chart inspector **22**
IfvCircularScaleDisplay class **222**
Ilv3dBarChartDisplay class **163**

IlvAbstractChartCursor class **224**
 drawGhost method **225**
 getLabel method **225**
 getValue method **225**

IlvAbstractChartData class **100, 101, 117, 126**
 addDataSet method **135**
 lock method **133**
 removeDataSet method **135**
 replaceDataSet method **135**
 setDataSet method **134, 135**

IlvAbstractChartDisplayer class **117, 150**
 getParentDisplayer method **154**
 setPalette method **154**

IlvAbstractChartFunction class **101, 128**

IlvAbstractGridDisplayer class **219**
 drawMinorLines method **221**
 setDrawOrder method **221, 225**

IlvAbstractProjector class **119**

IlvAbstractScaleDisplayer class **118, 194**
 drawLabelOnCrossings method **203**
 drawOverlappingLabels method **203**
 getCoordinateInfo method **212**
 setCrossingValue method **196**
 setDrawOrder method **197**
 setLabelLayout method **203**
 setRelativePosition method **196**
 setStepLabels method **201**
 setTickLayout method **202**
 setValueToLabelCB method **201**

IlvAffineChartTransformer class **213**

IlvBarChartDisplayer class **162**

IlvCallbackChartFunction class **101, 128, 129**

IlvCartesianChart class **101, 102, 118, 120**

IlvCartesianProjector class **119, 121, 221, 226**
 Orientation type **210**
 setOrientation method **212**

IlvChartCoordinateTransformer class **213**

IlvChartCrossHairInteractor class **231, 233**

IlvChartDataInteractor class **231**

IlvChartDataListener class **139**
 dataPointAdded method **139**
 dataPointChanged method **139**
 dataPointRemoved method **139**
 dataSetAdded method **139**
 dataSetChanged method **139**
 dataSetRemoved method **139**

IlvChartDataPointInfo class **185**

IlvChartDataSet class **100, 102, 117, 127**
 addPoint method **134**
 areListenersEnabled method **137, 139**
 enableListeners method **137**
 endBatch method **138, 245**
 insertPoint method **134**
 lock method **133**
 removePointAndInfo method **134**
 setPoint method **134**
 startBatch method **138, 245**
 unlock method **133**

IlvChartDataSetListener class
 dataPointAdded method **139**
 dataPointChanged method **139**
 dataPointRemoved method **139**

IlvChartDragPointInteractor class **231, 233**

IlvChartGraphic class **19, 101, 102, 117, 120**
 addAbscissaCursor method **228**
 addAbscissaGrid method **223**
 addOrdinateCursor method **228**
 addOrdinateGrid method **223**
 enableFastScroll method **244**
 getAbscissaInfo method **212**
 getData method **105, 131**
 getOrdinateInfo method **212**
 redrawWhenNotified method **246**
 setLegend method **219**
 setProjectHorizontally **157**
 setProjectVertically **157**
 setScrollMode method **239**
 setScrollRatio method **239**
 updateAndRedraw method **246**

IlvChartHighlightPointInteractor class **231, 234**

IlvChartInfoViewInteractor class **231, 234**

IlvChartInteractor class **230**

IlvChartInteractorManager class **236**

IlvChartLayout class **118, 146**
 getLayout method **148**
 setDataDisplayArea method **148**
 setDataDisplayAreaRelatively method **148**
 setGraphArea method **148**
 setGraphAreaRelatively method **148**

- setLayout method **148**
- IlvChartLegend class **19, 69, 216**
- IlvChartLegendItem class **217**
- IlvChartPanInteractor class **230, 233**
- IlvChartPointSet class **101, 127, 131**
- IlvChartScrollInteractor class **230, 232**
- IlvChartSelectInteractor class **231, 235**
- IlvChartYValueSet class **101, 127, 131, 168**
- IlvChartZoomInteractor class **230, 232**
- IlvCircularChartCursor class **226**
- IlvCircularGridDisplayer class **222**
- IlvCircularScaleDisplayer class **118, 226**
- IlvCircularScaleDisplayer class **205**
- IlvCompositeChartDisplayer class **171**
 - getDisplayer method **172**
 - setDisplayerFactory method **172**
 - setDisplayerModel method **172**
- IlvCoordinateInfo class **119**
 - getUserDataMax method **213**
 - getUserDataMin method **213**
 - setUserDataMax method **213**
 - setUserDataMin method **213**
 - setUserDataRange method **213**
- IlvCoordinateTransformer class **119**
- IlvDouble type **127**
- IlvDoublePoint class **101, 127**
- IlvGraphic class
 - getHolder method **243**
- IlvGraphicHolder class **243**
 - initReDraws method **243**
 - reDrawViews method **243**
- IlvGridDisplayer class **221**
- IlvHiLoBarChartDisplayer class **106, 166**
- IlvHiLoChartDisplayer class **164**
- IlvHiLoOpenCloseChartDisplayer class **175**
 - getHiLoDisplayer method **175**
 - getOpenCloseDisplayer method **175**
- IlvLabel class **169**
- IlvMarkedPolylineChartDisplayer class **173**
- IlvMarkedPolylineDisplayer class
 - getLineDisplayer method **173**
 - getMarkerDisplayer method **173**
- IlvMemoryChartData class **100, 101, 102, 117, 126**
- IlvMultiRectangularScaleDisplayer class **221, 222, 226**

- IlvPieChartDisplayer class **167**
- IlvPieChartGraphic class **168, 169**
- IlvPieSliceInfo class **169**
- IlvPointInfoArray class **185**
- IlvPointInfoCollection class **185**
- IlvPointInfoMap class **185**
- IlvPointInfoSingleton class **186**
- IlvPolarChart class **102, 118**
- IlvPolarProjection class **222**
- IlvPolarProjector class **119, 226**
 - getOrientedClockwise method **212**
 - setOrientedClockwise method **212**
- IlvPolygonChartDisplayer class **159**
- IlvPolylineChartDisplayer class **156, 192**
 - getProjectedPointsPalette method **157**
- IlvRadialChartCursor class **226**
- IlvRadialGridDisplayer class **222**
- IlvRectangularChartCursor class **226**
- IlvRectangularGridDisplayer class **221**
- IlvRectangularScaleDisplayer class **118, 122, 204, 221, 222, 226**
- IlvScatterChartDisplayer class **155**
- IlvScriptChartFunction class **101, 129**
- IlvShadowRectangle class **216**
- IlvSideBySideChartDisplayer class **181**
- IlvSimpleChartTransformer class **213**
- IlvSingleChartDisplayer class **154**
- IlvSingleScaleDisplayer class **197**
 - getStepLabelFormat method **201**
- IlvStackedChartDisplayer class **177**
- IlvStairChartDisplayer class **161**
- IlvStepChartDisplayer class **160**
- IlvXMLChartData class **248**
- IlvXMLPropertyReader class **251**
- IlvXMLValueListProcessor class **251**
 - initReDraws method
 - IlvGraphicHolder class **243**
- Insert icon **22**
- insertPoint method
 - IlvChartDataSet class **134**
- interactors
 - adding to chart interactor managers **236**
 - crosshair **233**
 - display-information **234**
 - drag-point **233**

- pan **233**
- scroll **232**
- select-data-points **235**
- select-data-points interactor **235**
- trigger-action **234**
- zoom **232**

L

- LabelLayout type **203**
- launching IBM ILOG Views Studio with Charts **16**
- layout object
 - description of **144**
 - general properties **146**
 - getting **148**
 - setting **148**
- layout objects
 - default instantiation **118**
 - description of **115, 118**
- Layout, Chart inspector page **29**
- legend
 - connecting to a chart **68**
 - parameters **35**
- Legend Inspector **34**
- legend items **217**
- legend objects
 - general properties **218**
- legends
 - adding **108, 112**
 - adding to a chart **219**
 - default instantiation **119**
 - description of **97, 115, 119, 217**
 - displaying **216**
- listeners
 - adding a data set listener object to a data set **142**
 - catching modifications **138**
 - description of **99, 135**
 - diagram of listener mechanism **136**
 - diagram of propagation process **138**
 - example of user-defined listener **140**
 - on chart data objects **139**
 - on data sets **139**
 - performing the updates **137**
 - propagation of modifications **138**
 - subclassing `IlvChartDataSetListener` **141**

- user-defined **140**
- lock method
 - `IlvAbstractChartData` class **133**
 - `IlvChartDataSet` class **133**
- lock/unlock system **133**
- logarithmic scales **66**
- logarithmic transformations **213**

M

- Main window **17, 62**
- major ticks **197**
- marked polyline displayer **173**
- marked polyline, displayer type **44**
- minor ticks **197**
- Miscellaneous, Chart Inspector page **31**
- modifying data
 - using batch operations **245**
- Move item up/Move item down icons **22**

O

- ordinate scales
 - and displayers **51**
 - customizing **49, 67, 107, 111**
 - Cartesian charts **46, 49**
 - polar charts **86**
 - related **52**
 - using several independent **39**
- orientation of scales **210**
- Orientation type
 - `IlvCartesianProjector` class **210**
- out-of-bounds data points **157**
- out-of-bounds points **191**

P

- palettes
 - defining **190**
- Palettes panel **17**
- pan interactor **233**
- performance when adding data points to charts **243**
- pie chart **20, 81**
 - creating **169**
- pie displayers **167**

- point information object
 - adding to a data point **188**
 - creating **187**
- polar charts
 - adding legends to **112**
 - creating **73, 109, 110**
 - customizing projection **85**
 - definition **72**
 - definition of **91, 97**
 - example of chart layout **145**
 - representing time values **81**
 - representing values expressed in radians
 - by applying a transformation **73**
 - by setting a starting angle and a range **79**
 - scales orientation **212**
 - simple example **109**
 - Temperatures Chart example **109**
 - types of grid displayers **222**
 - types of scale displayers **222**
- polygon displayer **159**
- polyline displayer **156**
- Polyline, displayer type **65, 66**
- polyline, displayer type **85**
- projecting
 - out-of-bounds data points **191**
- projection
 - customizing in polar charts **85**
 - types **26**
- Projection, Chart inspector page **26**
- projectors
 - creating **121**
 - default instantiation **119**
 - description of **115, 119**
 - diagram of mapping data points into screen coordinates **124**
 - projecting transformed data points **124**
 - setting **121**

R

- radians
 - converting to degrees **76**
 - representing on a polar chart **73**
- range
 - defining in a polar chart **79, 85**

- real-time data
 - displaying in charts **239**
- rectangular scale displayer **204**
- reDrawViews method
 - IlvGraphicHolder class **243**
- reDrawWhenNotified method
 - IlvChartGraphic class **246**
- Remove icon **22**
- removeDataSet method
 - IlvAbstractChartData class **135**
- removePointandInfo method
 - IlvChartDataSet class **134**
- replaceDataSet method
 - IlvAbstractChartData class **135**
- Resources panel **58, 68**

S

- scale displayers
 - adding to a chart **208**
 - class hierarchy **195**
 - default instantiation **118**
 - description of **115, 118**
 - general properties **195**
 - single
 - circular **205**
 - general properties **198**
 - rectangular **204**
 - types for Cartesian **221**
 - types for polar charts **222**
- scales
 - adding cursors **226**
 - adding grid displayers **221**
 - changing orientation **210**
 - circular scales **92**
 - converting data values to step labels **201**
 - customizing **106, 107, 111, 210**
 - customizing abscissa **113**
 - customizing ordinate **113**
 - defining drawing order **197**
 - defining minimum and maximum values **212**
 - defining position of labels **203**
 - defining step labels **201**
 - defining steps and substeps **200**
 - defining the abscissa **121**

- defining the ordinate **122**
- defining the position of **196**
- defining the position of ticks **202**
- drawing overlapping step labels **203**
- drawing scales **194**
- fixing to a data value **196**
- fixing to a position **196**
- single
 - description of **197**
 - diagram of components **197**
 - step labels drawn at axes crossing **203**
 - types of graduation **92**
- Scales, Chart inspector page **27, 44, 67**
- scatter displayer **155**
- Scatter, displayer type **43, 44**
- screen coordinates
 - as projected by projectors **124**
 - description of **123**
- Script Editor **62**
- script function
 - creating a data set object **129**
- scroll interactor **232**
- scroll mode
 - fast **243**
 - stop **238**
- scroll modes
 - cyclic **239**
 - description of **238**
 - example of using **240**
 - setting **241**
 - shift **239**
- scroll ratio
 - setting **242**
- setCrossingValue method
 - IlvAbstractScaleDisplayer class **196**
- setDataDisplayArea method
 - IlvChartLayout class **148**
- setDataDisplayAreaRelatively method
 - IlvChartLayout class **148**
- setDataSet method
 - IlvAbstractChartData class **134, 135**
- setDisplayerFactory method
 - IlvCompositeChartDisplayer class **172**
- setDisplayerModel method
 - IlvCompositeChartDisplayer class **172**

- setDrawOrder method
 - IlvAbstractGridDisplayer class **221, 225**
 - IlvAbstractScaleDisplayer class **197**
- setGraphArea method
 - IlvChartLayout class **148**
- setGraphAreaRelatively method
 - IlvChartLayout class **148**
- setLabelLayout method
 - IlvAbstractScaleDisplayer class **203**
- setLayout method
 - IlvChartLayout class **148**
- setLegend method
 - IlvChartGraphic class **219**
- set-of-points data set
 - description of **127**
- set-of-values data set
 - description of **127**
- setOrientation method
 - IlvCartesianProjector class **212**
- setOrientedClockwise method
 - IlvPolarProjector class **212**
- setPalette method
 - IlvAbstractChartDisplayer class **154**
- setPoint method
 - IlvChartDataSet class **134**
- setProjectHorizontally method
 - IlvChartGraphic class **157**
- setProjectVertically method
 - IlvChartGraphic class **157**
- setRelativePosition method
 - IlvAbstractScaleDisplayer class **196**
- setScrollMode method
 - IlvChartGraphic class **239**
- setScrollRatio method
 - IlvChartGraphic class **239**
- setStepLabels method
 - IlvAbstractScaleDisplayer class **201**
- setTickLayout method
 - IlvAbstractScaleDisplayer class **202**
- setting
 - interactors **236**
 - scroll mode **241**
 - scroll ratio **242**
- setUserDataMax method
 - IlvCoordinateInfo class **213**

- setUserDataMin method
 - IlvCoordinateInfo class **213**
- setUserDataRange method
 - IlvCoordinateInfo class **213**
- setValueToLabelCB method
 - IlvAbstractScaleDisplayer class **201**
- side-by-side displayer **181**
- Specific, Legend inspector page **35**
- stacked 3D bar chart displayer **177**
- stacked bar chart displayer **177**
- Stacked bars, displayer type **57**
- stacked displayer **177**
- stacked polygon chart displayer **177**
- stacked/side-by-side chart, creating **56**
- stair displayer **161**
- startBatch method
 - IlvChartDataSet class **138, 245**
- starting angle
 - defining in a polar chart **79, 85**
- step displayer **160**
- step labels **197**
 - defining **201**
 - defining position **203**
 - drawing at axes crossing **203**
 - drawing overlapping labels **203**

T

- TickLayout type **202**
- ticks
 - defining **202**
- Ticks display, Scales subpage **29**
- Ticks, Scales subpage **28, 50, 55**
- time values
 - representing on a polar chart **81**
- transformation, radians to degrees **76**
- Transformation, Scales subpage **54, 67**
 - definition **28**
- transformations
 - applying to data values **213**
- transformers
 - affine **213**
 - simple **213**
- trigger-action interactor **234**

U

- unlock method
 - IlvChartDataSet class **133**
- updateAndReDraw method
 - IlvChartGraphic class **246**
- updating charts
 - description of **135**
 - releasing automatic updates **246**

Z

- zoom interactor **232**

