IBM ILOG JViews Charts V8.6

Introducing JViews Charts

Copyright

Copyright notice

Trademarks

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

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

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

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

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

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

Notices

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

Table of contents

Introducing IBM® ILOG® JViews Charts

Introduces the main features of IBM® ILOG® JViews Charts and indicates some typical uses.

In this section

What is a chart
Explains what chart represents and the various types of chart.

Static and dynamic charts
Explains what static and dynamic charts are.

Main features of a chart
Describes the different chart features available with JViews Charts.

Typical uses of charts
Describes the typical uses of a chart.

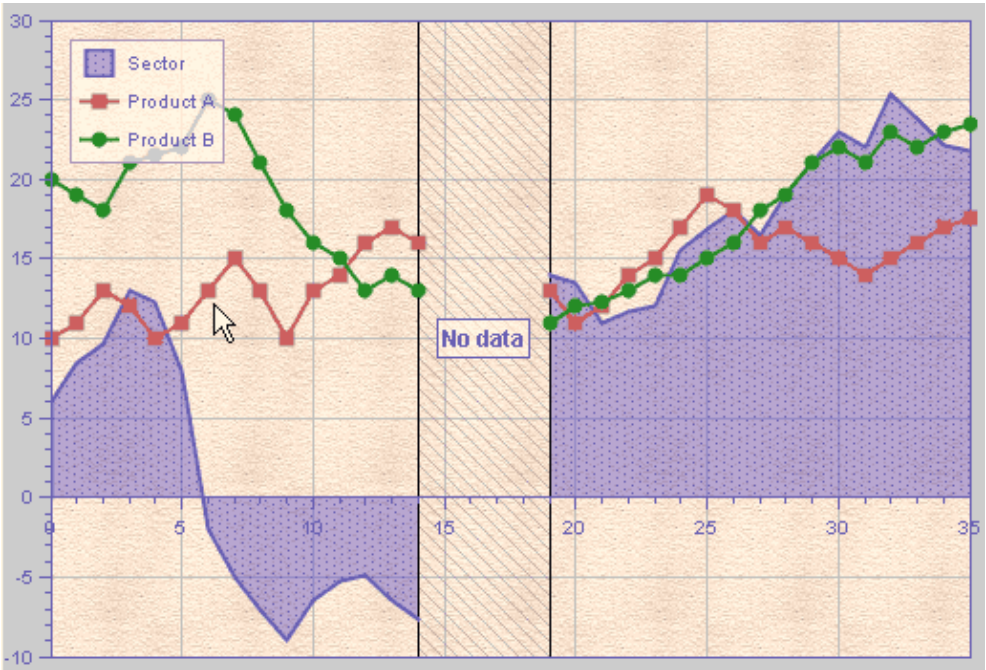
What is a chart

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.

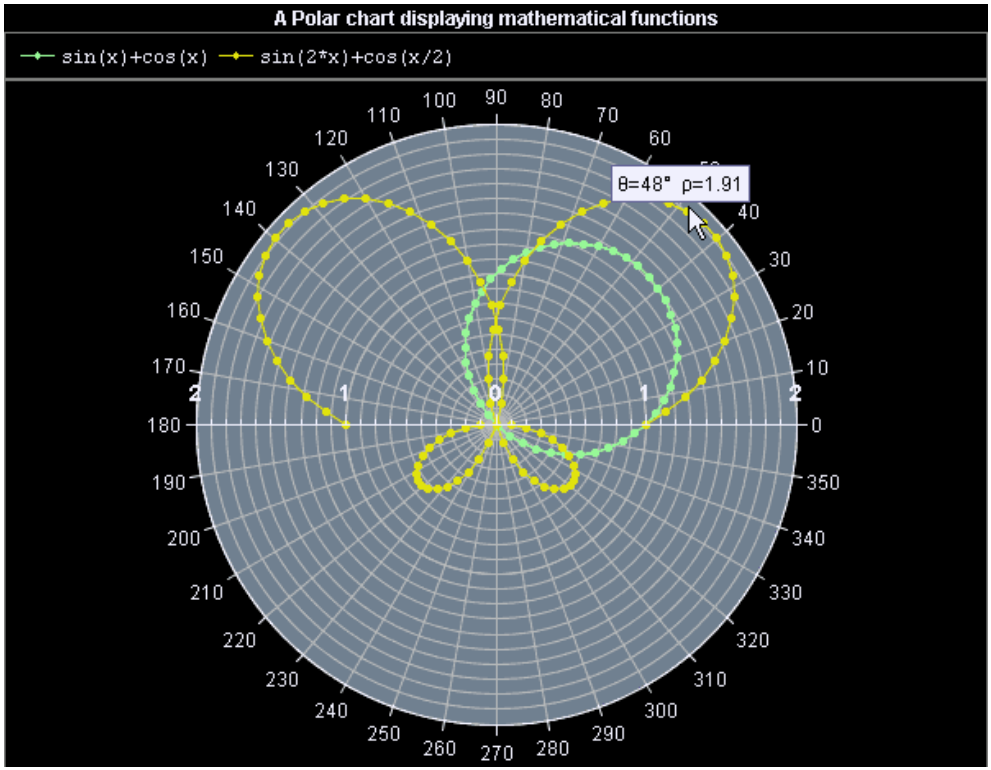
JViews Charts allows you to display data in charts that can be customized in various ways, and to interact with these charts in different manners. JViews Charts has been designed for optimum performance coupled with a clean object architecture that makes it the best solution for handling large and/or dynamic data models such as real-time supervision system applications.

JViews Charts provides a wide range of displays:

- 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.

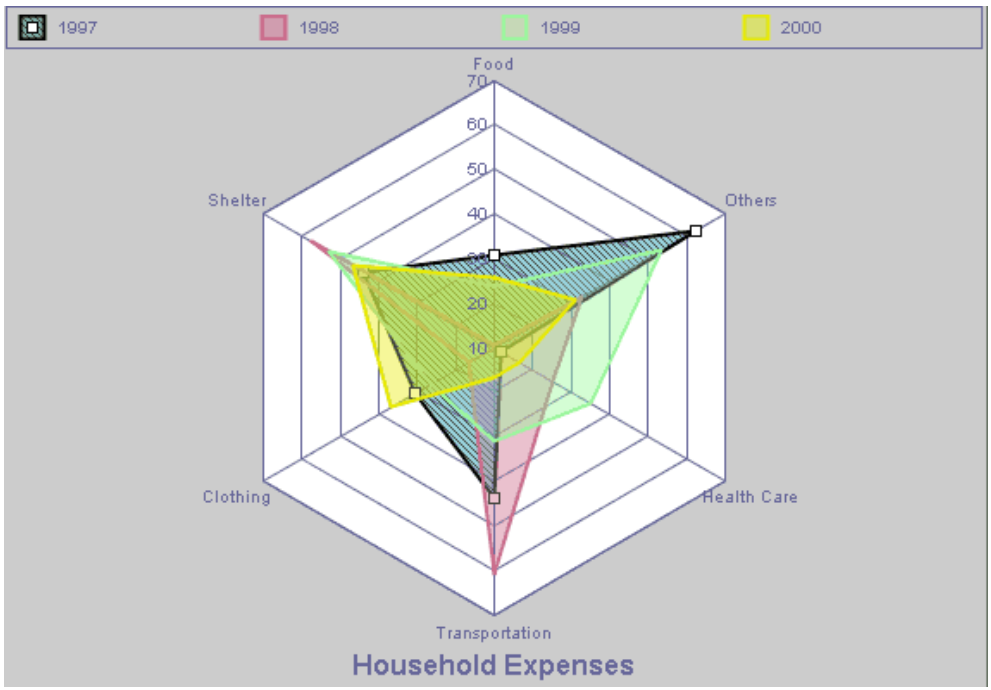


- Polar charts represent data in a circular way. The data is expressed using a polar system of coordinates (ᑭ, ϴ). The abscissa values are plotted along a circular scale. The ordinate scale, along which the -coordinates are plotted, is rectangular and is displayed radially.

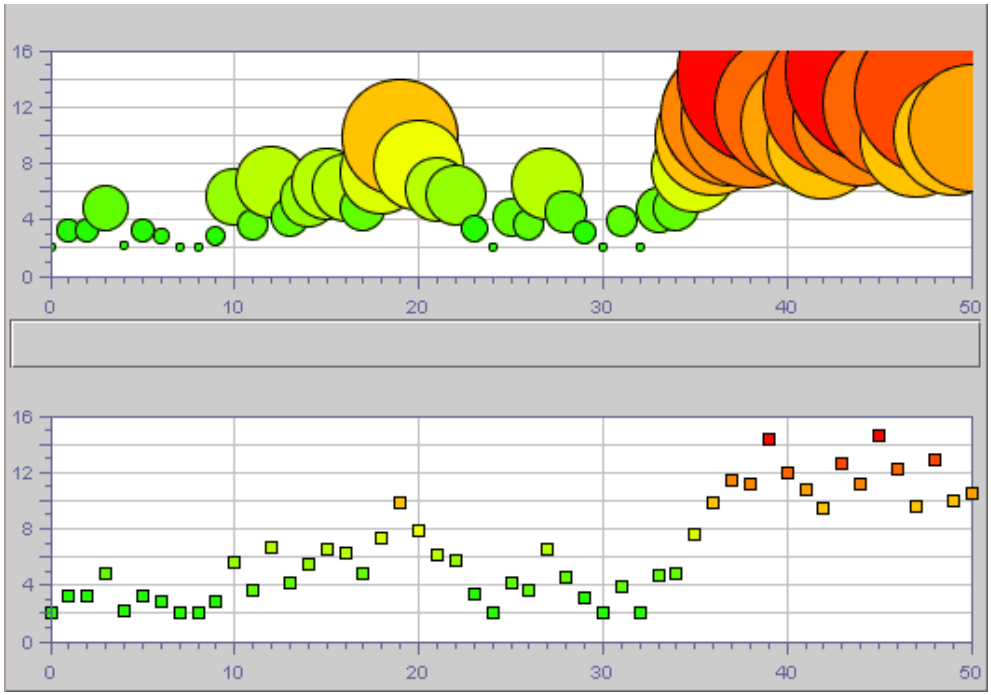


♦ Radar charts.

A radar chart allows us to see data series relationships and make comparisons based on multiple categories. In a radar chart, each category of values has its own axis radiating from a center point. Lines connect all the values in the same category series.



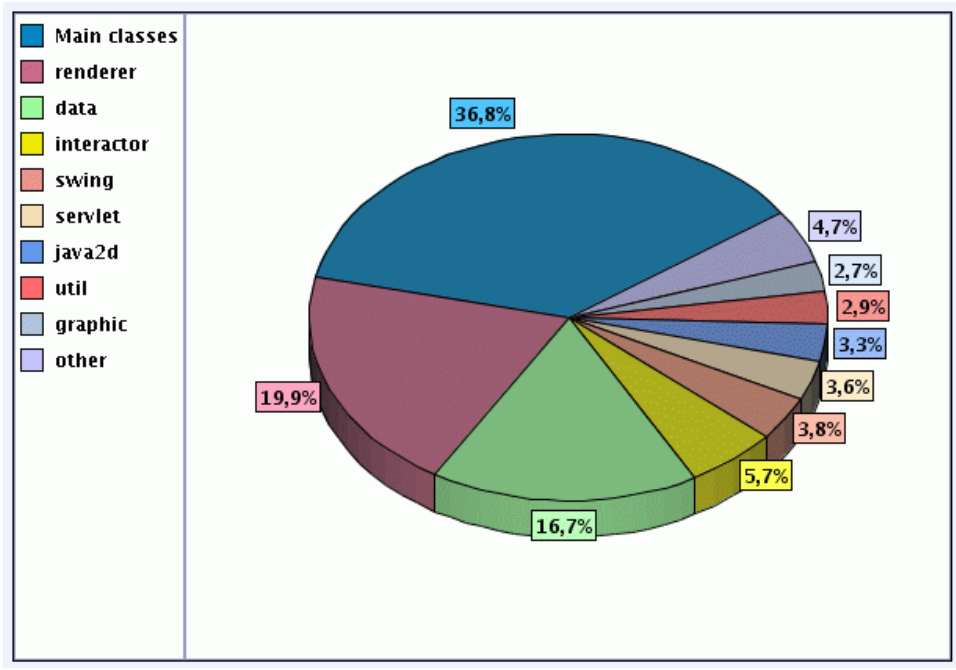
♦ Bubble charts.



♦ High/Low Charts.

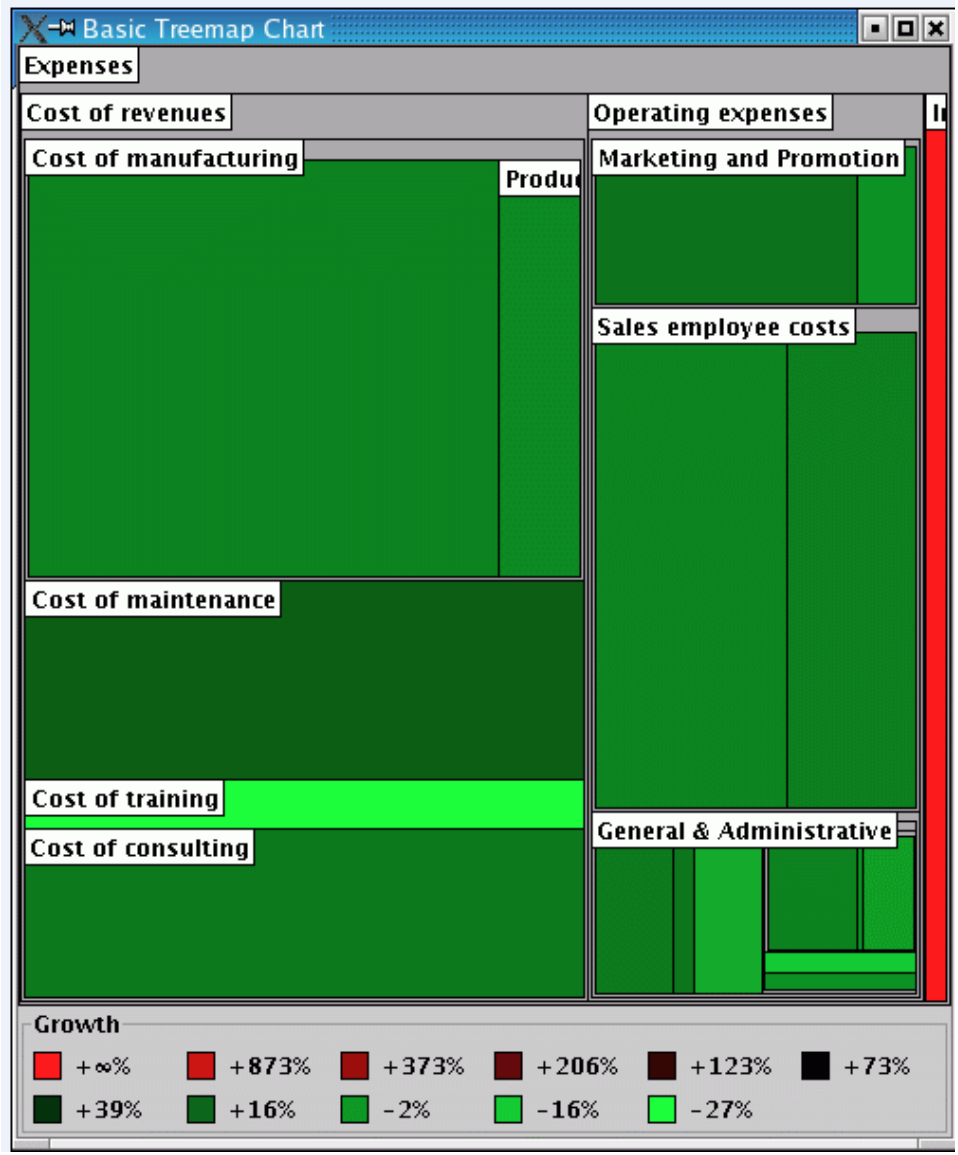
♦ Pie Charts.

A pie chart is used to represent percentages and proportions with which a whole is composed from parts.



♦ Treemap Charts.

A treemap chart is like a multiple level pie chart. It allows you to see important elements and hides unimportant details.


Basic Treemap Chart
Expenses
Cost of revenues
Operating expenses
Cost of manufacturing
Marketing and Promotion
Sales employee costs
Cost of maintenance
Cost of training
Cost of consulting
General & Administrative
Growth
+∞%
+873%
+373%
+206%
+123%
+73%
+39%
+16%
-2%
-16%
-27%

Static and dynamic charts

A chart can be static, in the sense that there are no changes in its appearance while it is displayed, or it can be dynamic, reacting to user actions or external data feeds or both.

A static chart, as the name implies, will not change once it is drawn: it is a snapshot of a given system. Report generating softwares allow you to create such static charts. A static chart is often created for documentation purposes. Examples include electronic schematics and org charts.

A dynamic chart, on the other hand, remains in contact with business data during the display phase and is expected to change over time in response to business-related changes. Examples of such changes are: values of new data points, an operator moves an object to a different position, and so on. On a computer display, as part of a larger computer system, the chart is connected to the underlying business objects by specialized software. As the business objects evolve, the elements of the chart automatically evolve with them. The rendering of the graphic objects that represent each business object is modified to show changing business conditions. This capability makes the chart a living entity, aware of changes in the underlying data, in fact, the term data-aware is often used to describe a display that is connected to business data in this way.

Main features of a chart

JViews Charts offers you the following features:

- Full-featured API. Dedicated classes that allow you to display your data in charts. These charts can be customized in various ways, and you can interact with these charts in different manners.
- Optimum performance coupled with a clean object architecture that makes it the best solution for handling large and/or dynamic data models such as real-time supervision system applications.
- Wide range of chart displays: Polyline, Bar, Area, Bubble, High-Low, Scatter, Stair, Combo, Pie, Treemap.
- Easy customization.
- A clear separation between the data and the graphical representations of the data.
- Charts are *data-aware*. Changes made to data are automatically reflected in the charts that display this data. The possible modifications made by interacting with a chart are also automatically reflected on the data.
- Dynamic control of the chart appearance by means of cascading style sheets (CSS).

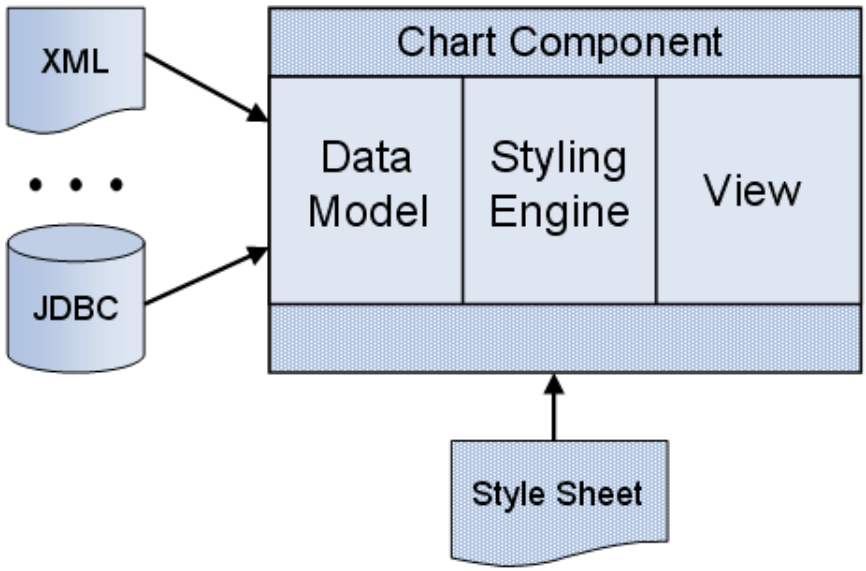
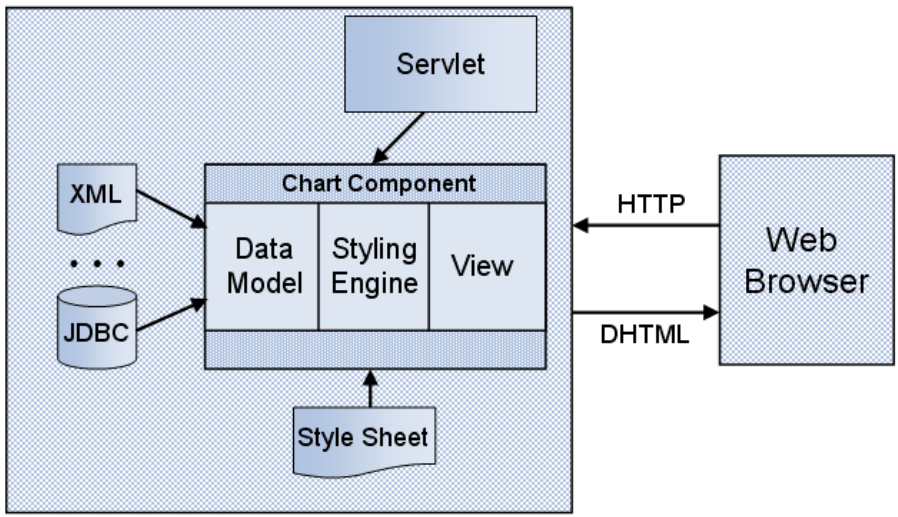


Chart Component Architecture

- Integration in any Swing-enabled Java™ application as a classic JComponent.

- Integration in Eclipse® /RCP applications as an SWT control.
- A set of JavaBeans™ that can be used within your favorite IDE (Integrated Development Environment).
- Coordinate transformation along a given axis. The transformation can be either linear or non-linear, which allows you to implement features such as logarithmic axis or local zoom.
- Load On Demand

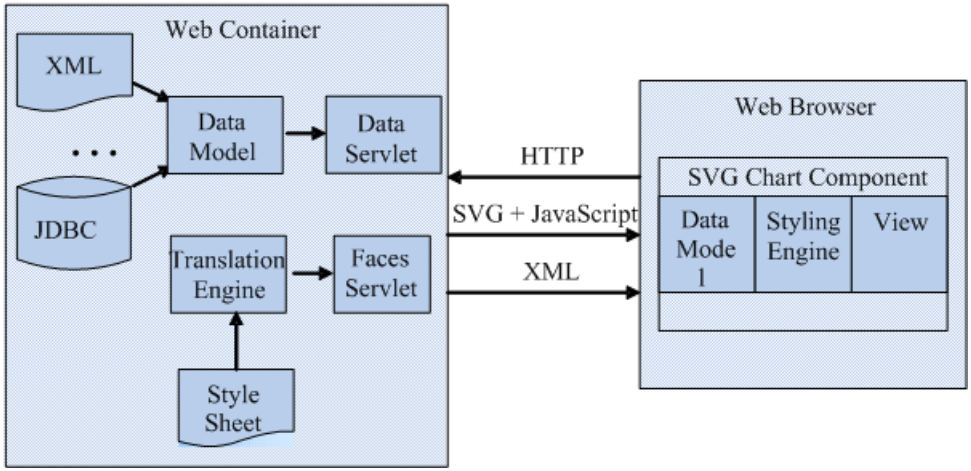
  The load-on-demand mechanism allows you to connect the charts to very large data sets by providing a control on the memory usage by loading only the values that need to be displayed.
- Thin-Client architecture
  - A set of classes based on the standard Servlet Java technology
  - Built-in support for image generation
  - JPEG and PNG image output formats natively supported (additional formats can be added through custom encoders)
  - Automatic client-side image map generation
  - JavaServer™ Faces tag library for the development of server pages



Thin-Client Chart Application Architecture

- Rich Web Client
  - JavaServer Faces tag library for the development of server pages
  - Client-side dynamic rendering in SVG

- Ajax architecture
- Server-side scalability



Rich Web Charts Application Architecture

♦ Customizers. These are predefined Swing components that can be used to change particular aspects of the style sheet of a chart component.

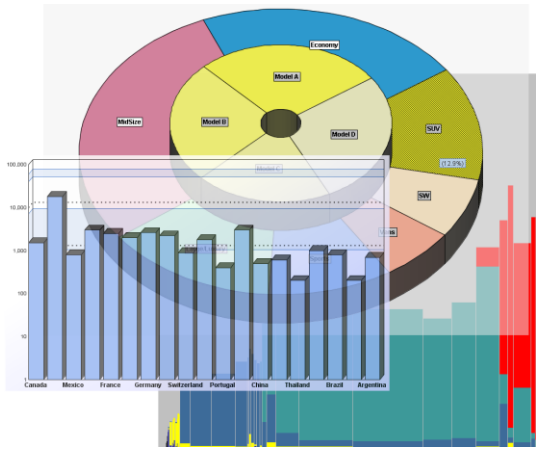
♦ Full Printing API

- Based on the standard Java 2 Printing API
- Multiple-page printing
- Rich page format (paragraph alignment, local fonts, and so on)
- Composite document (Chart, table, text in one document)
- Extensible framework

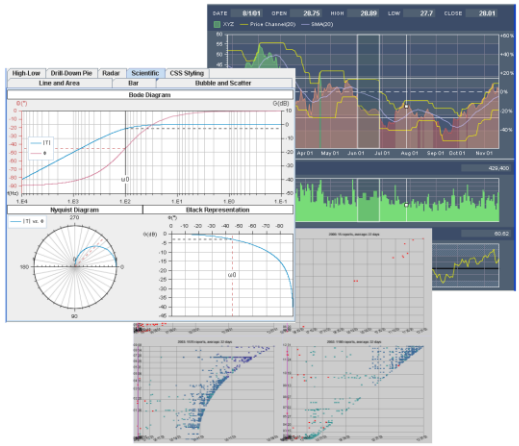
Typical uses of charts

You can use IBM® ILOG® JViews Charts to report data to communicate their values and trends, for example.

- Applications that need to report data to communicate their values and trends. The audience can be large (for instance Nasdaq evolution displayed in a search-engine portal), or it can be restricted to some employees of a company (financial dashboard dedicated to the top level management, application performance dashboards for MIS, Web dashboards for web team). These applications use classical charts that do not require any training (bar graphs, pie charts). These charts are read-only and often published on the web (thin client), and these applications are used occasionally.



- Mission-critical applications that are used by professionals to perform their daily job. These charts are often changed in real time, that is, they are connected to the data stream and updated in real time. These charts are customized for the application need and require precise strategies for scrolling, zooming-in, spotting data of interest, interacting and editing data. Such applications exist in all industries: applications for traders, testbeds for engines (cars, airplanes, and so on) and other machines, network or application management, scientific research.



You can find more examples of charts in the **samples page**.

Basic Concepts

Introduces you to the generic concepts of IBM® ILOG® JViews Charts.

In this section

Clear distinction between data and display
Explains the architecture IBM® ILOG® JViews Charts is based on.

Types of chart
Describes the different types of chart offered by IBM® ILOG® JViews Charts.

Supported graphical representations
Describes the graphical representations supported by IBM® ILOG® JViews Charts.

2-D versus 3-D
Describes the 3-D option.

Chart area
Explains what a chart area is.

Header and Footer
Explains what headers and footers are.

Axis
Explains what an axis represents on a chart.

Scales
Explains what scales are.

Legend
Explains what a legend is.

Grids
Explains what a grid is.

Decorations
Describes the different decorations available with JViews Charts.

Drawing Order
Explains what the drawing order consists of.

Interactors
Explains what the interactors are.

Predefined Interactors
Lists the predefined interactors available with JViews Charts.

Clear distinction between data and display

IBM® ILOG® JViews Charts is based on the *Separable Model Architecture*. This architecture is a variant of the Model-View-Controller model that was introduced in Swing. In this design, the model manages the data or the values represented by the component, while the view manages the graphical representation of the model, and handles interaction on it.

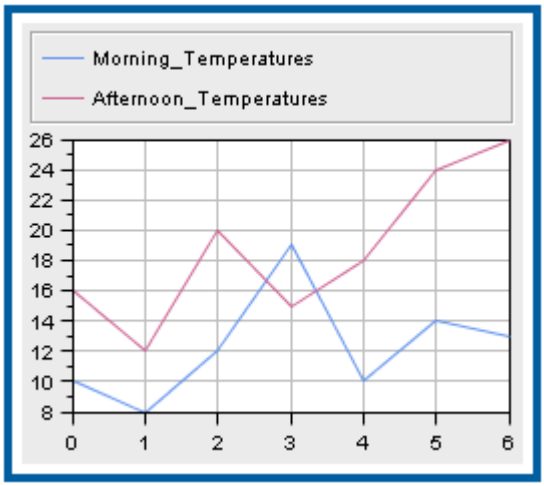
The use of this design allows you to have a clear distinction between:

- the *chart data model* that handles the sets of data by means of data sources (or *data sets*), and
- the *chart renderers* that draw the graphical representation of the data.

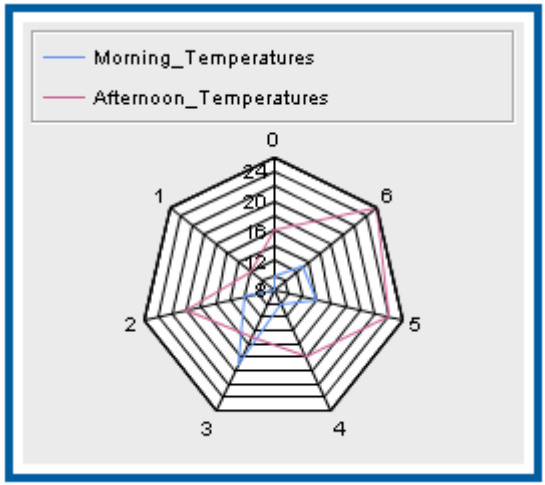
Types of chart

JViews Charts proposes five different types of chart:

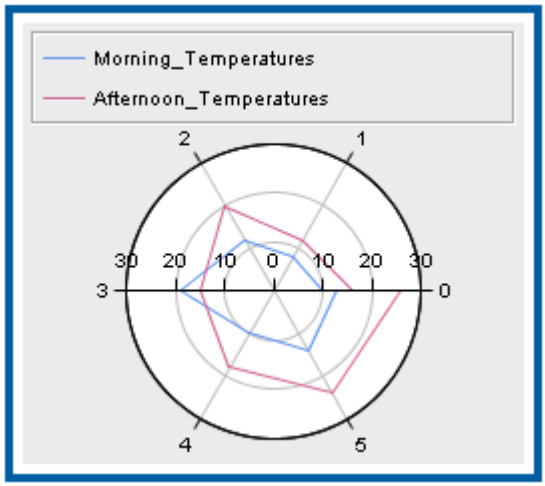
- Cartesian chart



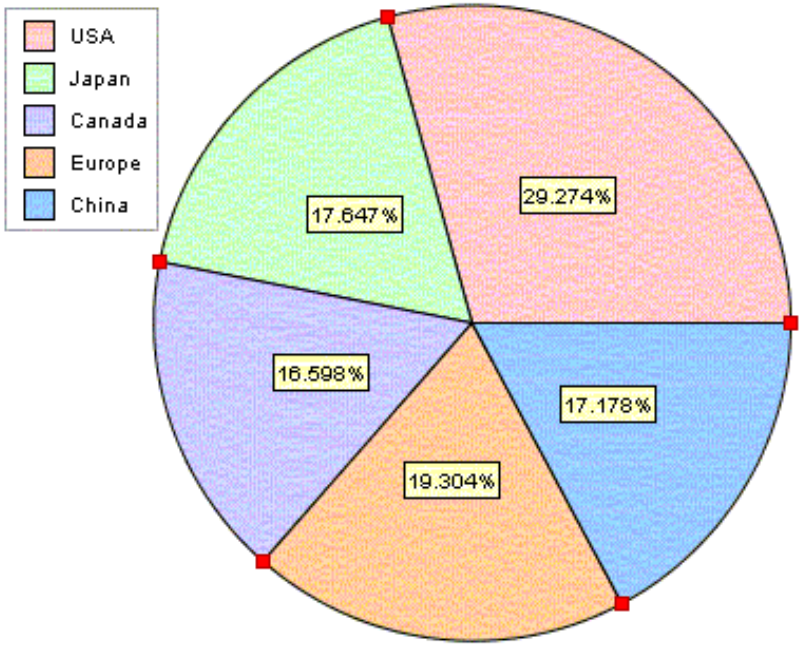
- Radar chart



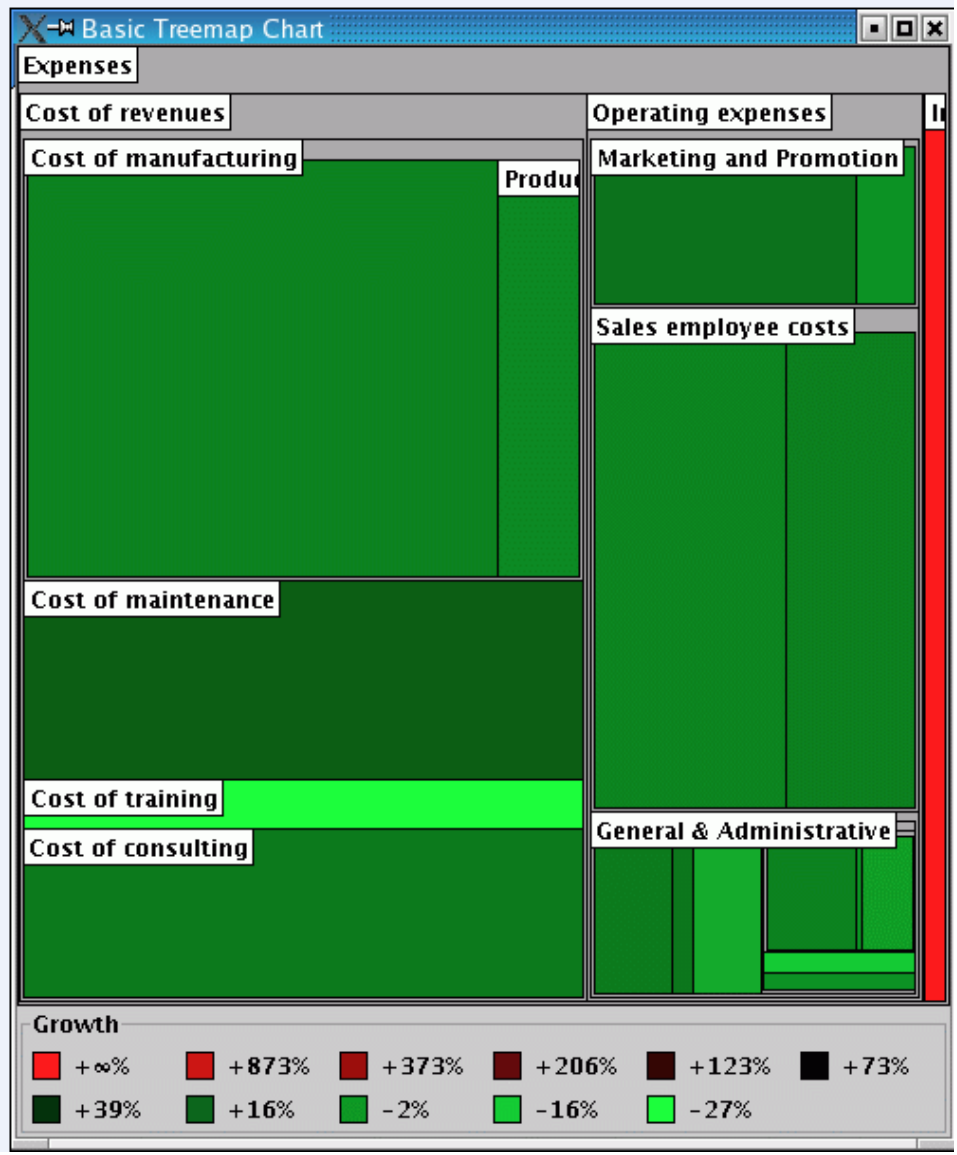
- Polar chart



- Pie chart



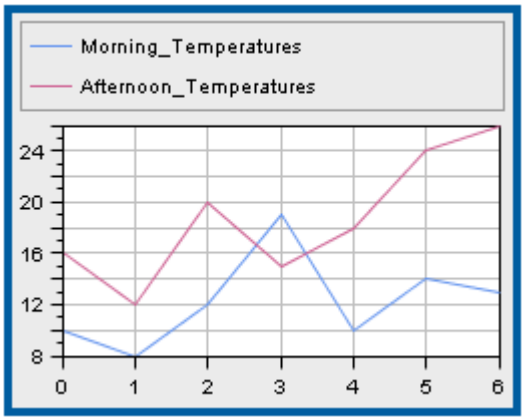
- Treemap chart


Basic Treemap Chart
Expenses
Cost of revenues
Operating expenses
Cost of manufacturing
Produ
Marketing and Promotion
Sales employee costs
Cost of maintenance
Cost of training
Cost of consulting
General & Administrative
Growth
+∞%
+873%
+373%
+206%
+123%
+73%
+39%
+16%
-2%
-16%
-27%

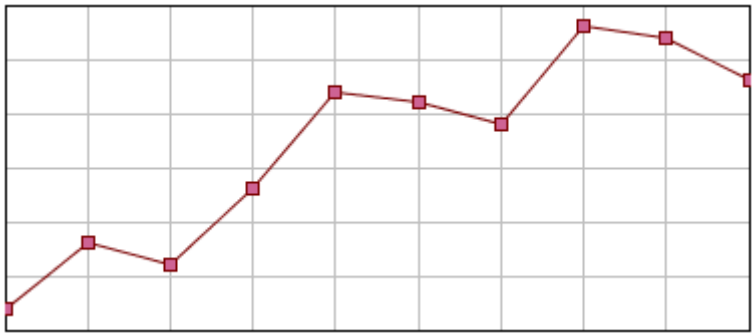
Supported graphical representations

JViews Charts supports ten different types of graphical representation:

- Polyline

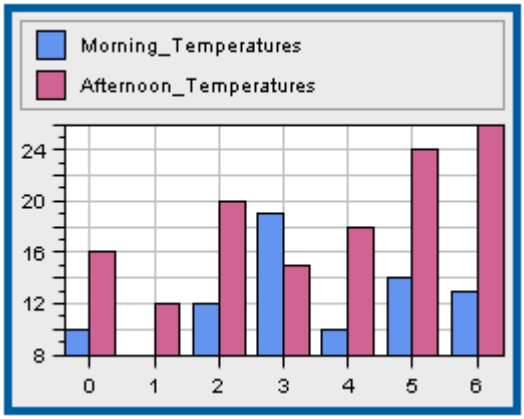


A single polyline renderer can draw an additional marker for each data point, for example a square marker, as illustrated below:

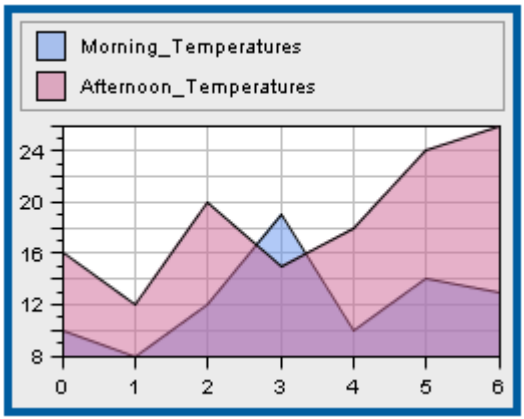


By default, the style of the marker is computed according to the style of the renderer, but you can also specify a rendering style of the marker symbol.

- Bar

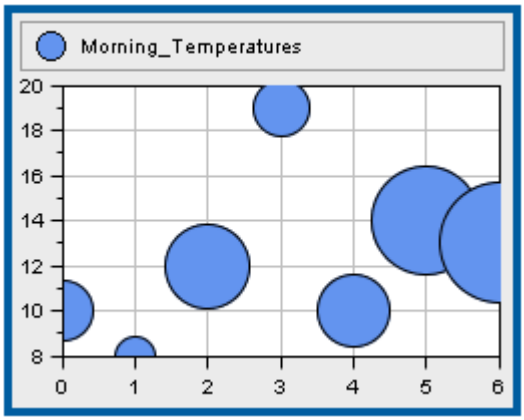


- Area



- Bubble

  A bubble chart represents a two-dimensional data model as bubbles of variable size. The data model should be described by two data sets, the first data set determining the location of the bubbles, and the second data set determining the size of the bubbles.



- High-Low

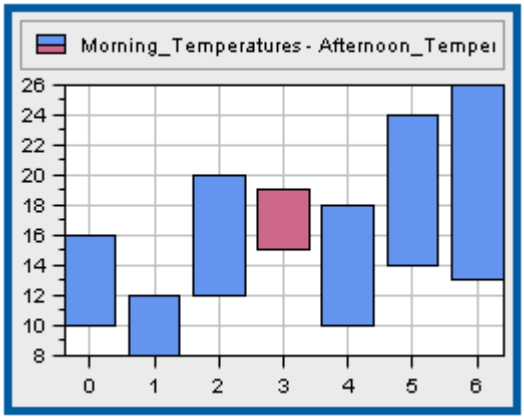
  The High-Low graphical representation renders two data sets with first and second items and defines two rendering styles:

  - rise style

    Used to draw the high-low items for which the corresponding first value is less than the second value.

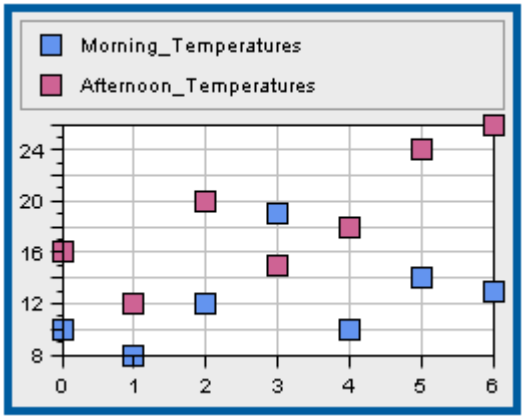
  - fall style

    Used to draw the high-low items for which the corresponding first value is greater than the second value.



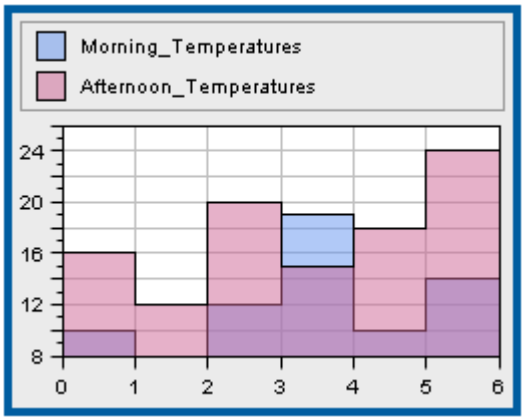
- Scatter

  The Scatter render represents a data set with scattered graphical markers.

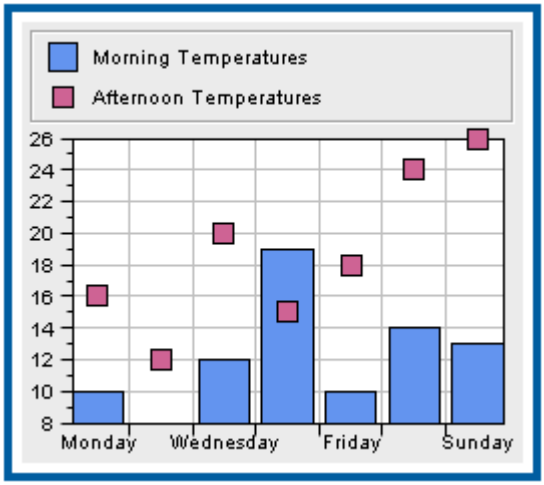


- Stair

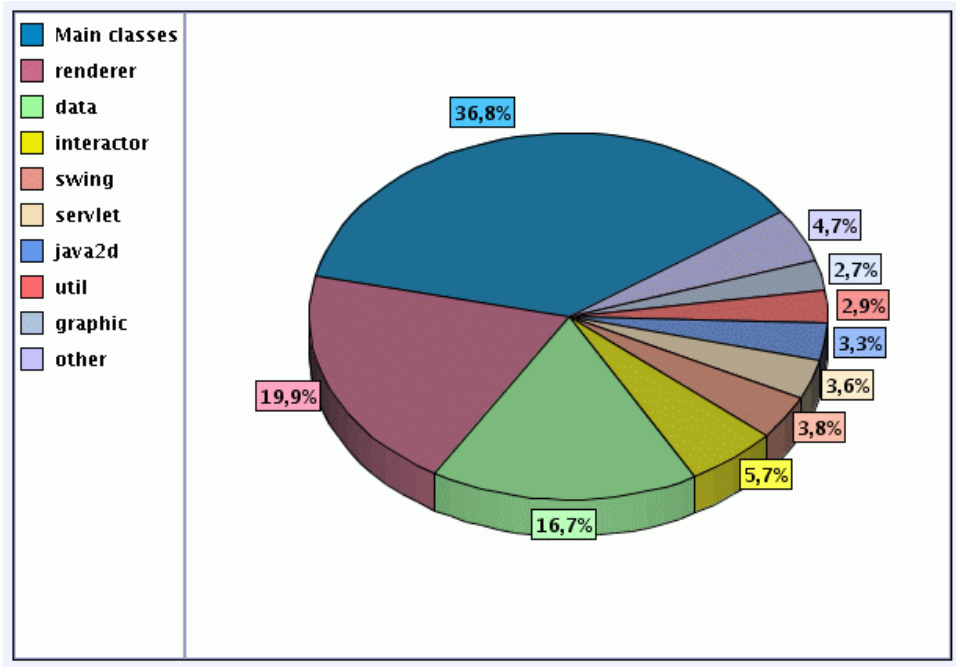
  The Stair renderer represents a transition between two values as a stair instead of straight lines.



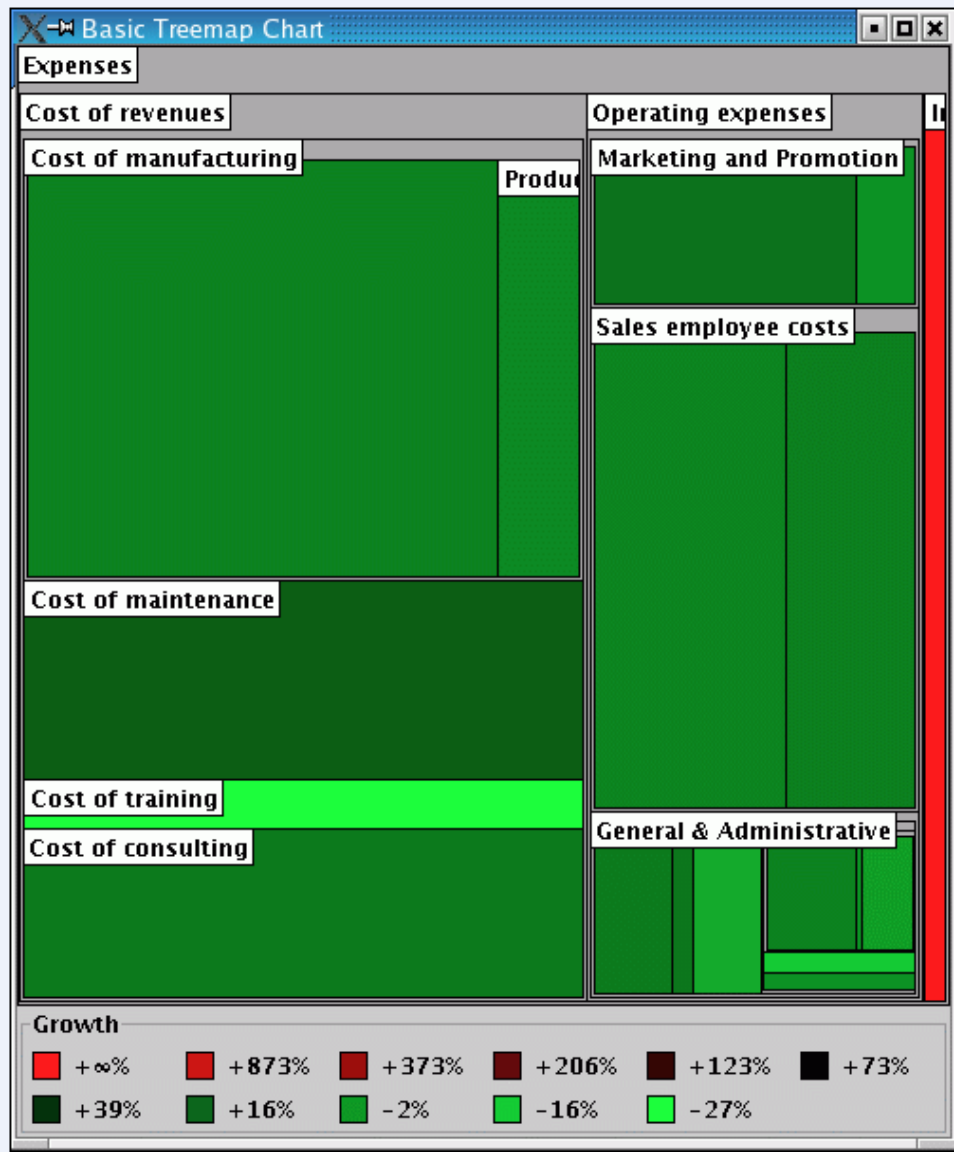
- Combo


Morning Temperatures
Afternoon Temperatures
26
24
22
20
18
16
14
12
10
8
Monday
Wednesday
Friday
Sunday

♦ Pie


Main classes
renderer
data
interactor
swing
servlet
java2d
util
graphic
other
36,8%
19,9%
16,7%
5,7%
3,8%
3,6%
3,3%
2,9%
2,7%
4,7%

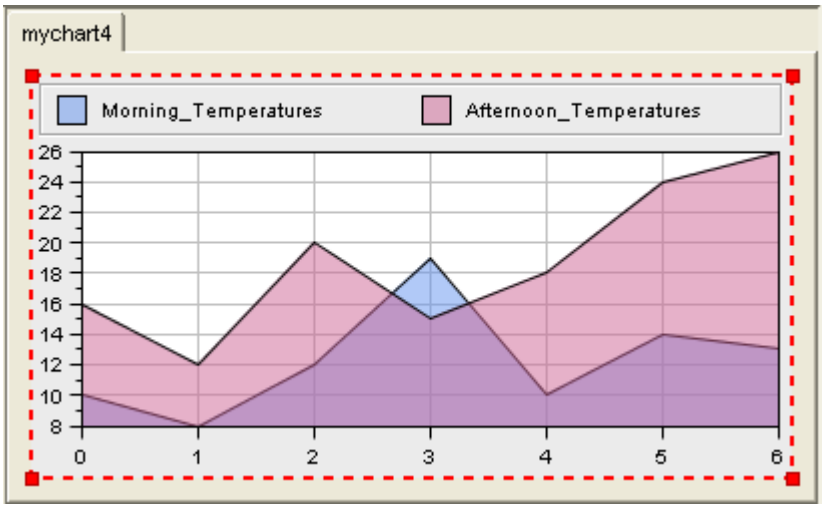
♦ Treemap


Basic Treemap Chart
Expenses
Cost of revenues
Operating expenses
Cost of manufacturing
Produ
Marketing and Promotion
Sales employee costs
Cost of maintenance
Cost of training
Cost of consulting
General & Administrative
Growth
+∞%
+873%
+373%
+206%
+123%
+73%
+39%
+16%
-2%
-16%
-27%

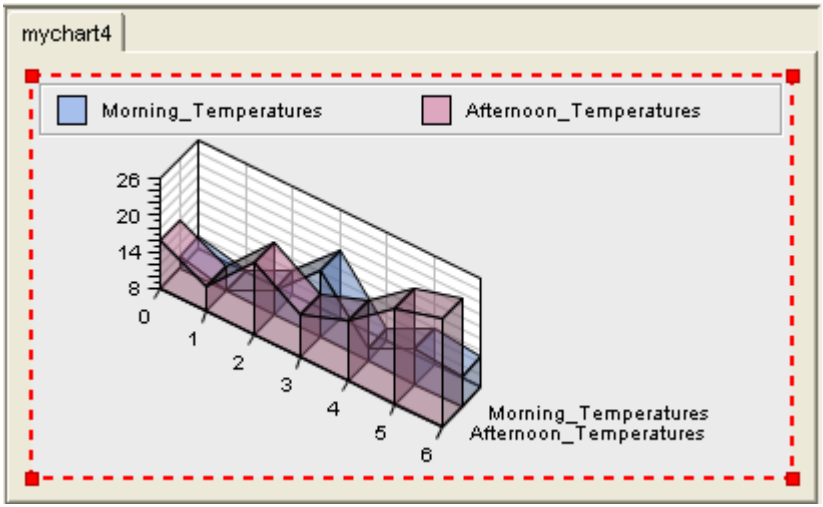
2-D versus 3-D

JViews Charts provides the ability to display a two-dimensional data model using three-dimensional rendering. Only Cartesian and Pie charts support 3-D rendering. If you try to switch from a 2-D view to a 3-D view on a chart that does not support this mode (for example, a Radar chart) you will not get any error. The visual appearance of the chart will not change.

You can easily switch between a 2-D and a 3-D display, as illustrated below:



Two-Dimensional Display



Three-Dimensional Display

As you can see, the structure of the chart is not altered. The chart components like header, footer, legend, renderers, scales, grids, decorations, and so on, remain unchanged.

Chart area

The chart area is the place where all the drawing operations are performed (the graphical representations of the data themselves and the decorations).

Within the chart area component, you can distinguish two areas:

- a drawing rectangle, where drawings are performed (both charts and scales).

  All the drawings occurring in the chart area are clipped by this rectangle.

- a plotting rectangle, the region where data is projected and displayed.

  The bounds of this rectangle are computed according to the drawing rectangle and the internal margins of the chart area.

These areas are computed by the chart area component through a special layout manager, either automatically or according to the user's preferences. To manually modify the plotting rectangle bounds, you need to change one or several margins (left, right, top, or bottom) that are expressed relative to the drawing rectangle. See *Chart Components*.

Header and Footer

These components are both optional components and are added to the chart either on top of or at the bottom of the chart area, respectively.

For example, these components can be used to set the chart title or to contain a GUI panel. See *Chart Components*.

Axis

A chart is composed of:

- Exactly one abscissa axis.
- One or several ordinate axes.

The axes are automatically created by a chart, which uses by default only one y-axis. The first *y-axis* is also referred to as the *main ordinate axis*. Other *y*-axis can be added to a chart. See *Chart Components*.

Scales

Scales might be defined as a graphical representation of a chart axis and are automatically configured by a chart when it is created.

A scale is composed of the following elements:

- An axis representation, which depends on the chart projection (could be a line or an arc).
- Major ticks, the marks drawn on the axis at each step of the scale.
- Steps 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.
- A title, which can be placed anywhere along the axis representation.

Depending on the type of chart, scales can be:

- Rectangular (that is, axes that are parallel to the x- and y-axes of the screen).
- Circular (generally used in Polar charts).

See *Chart Components*.

Legend

The legend component is an optional component that can be added either to the chart itself or to another container.

A legend is associated with a chart and updated automatically when needed.

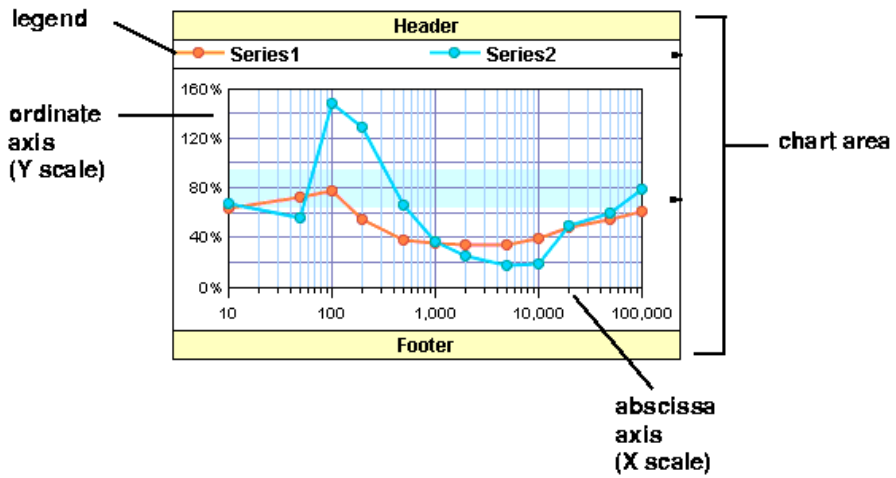


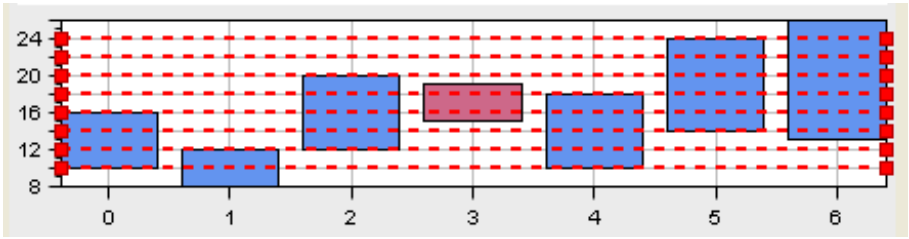
Chart Components

Grids

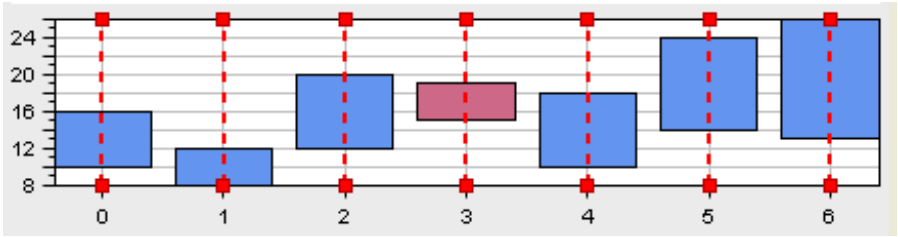
Grids help you locate the data points on a chart.

A grid is a graphical indicator of data values. A grid is attached to an axis and is composed of major gridlines and minor gridlines.

- Horizontal grids



- Vertical grids



Decorations

A chart can have the following decorations:

- Data indicators

  Data indicators that graphically represent the values of data points or a given data interval in the chart area. For example, you can decorate the chart by highlighting the week-end period using a data indicator which represents a data interval equal to the week-end period. The indicators are of the following type:

  
  A range indicator along the x-axis.

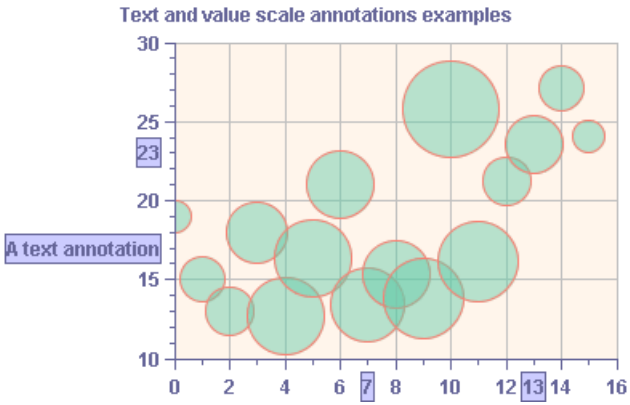
  
  A range indicator along the y-axis.

  
  A value along the x-axis.

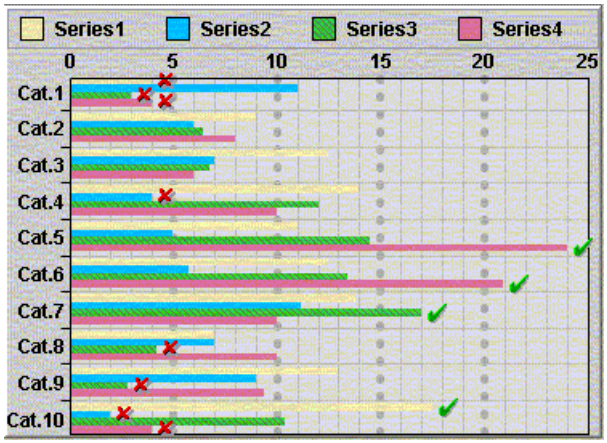
  
  A value along the y-axis.

  
  A data window.

- Annotations that display the values on a scale.



- Labels linked to given data points.



♦ An image within the plotting area of the chart.

An image can be drawn according to three different modes:

- TILED: The image is drawn as a replicated pattern in the plot area.
- SCALED: The image is scaled so that it fills the plot area.
- ANCHORED: The image is drawn at a fixed position.

Decorations are drawn according to a *drawing order*. The drawing order lets you control the position of a given decoration in the drawing queue of a chart.

Decorations are handled as an ordered list according to the decorations drawing order: decorations with the lowest drawing order are drawn first, decorations with the highest drawing order are drawn last.

Drawing Order

The drawing order lets you control the position of a given decoration in the drawing queue of a chart.

The drawing order also defines whether a decoration should be drawn above or below the graphical representations of the chart data: decorations with a negative drawing order are drawn below the chart representations, while decorations with a zero or positive drawing order are drawn above the chart representations.

Interactors

Interactors let you associate one or several behaviors to a chart object.

The following ready-to-use interactors are available:

- A zoom interactor to zoom in and out within the displayed data.
- Two scroll interactors to scroll the displayed data with the arrow keys along a given axis.
- A pan interactor to scroll the displayed data with the mouse.
- A set of interactors to modify the properties of a local zoom axis transformer.
- Data interactors that allow the user to do the following:
  - Edit a data point
  - Highlight or display information related to a data point
  - Select data points

The interaction mechanism implemented in the JViews Charts library provides a notification process that interactors might use to notify listeners when interactions have been performed. This notification mechanism is used by most of the default interactors previously listed and hence allows you to easily add a custom behavior to the predefined interaction without the need to subclass.

Predefined Interactors

Type of Interactor	Description
Zoom interactor	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.
X-Scroll Interactor	Lets the user scroll through the displayed data along the x-axis.
Y-Scroll Interactor	Lets the user scroll through the displayed data along the y-axis.
Pan Interactor	Lets the user scroll through the displayed data by dragging the mouse in any direction.
Action Interactor	Lets the user execute an action when a specific key is pressed.
Local Pan Interactor	Lets the user scroll the zoomed data window by dragging the mouse in any direction.The interaction starts when the user clicks within the zoomed data window.
Local Reshape Interactor	Lets the user reshape the zoomed data window by dragging the zoomed area bounds when the mouse moves over one of the zoomed data window bounds.
Local Zoom Interactor	Lets the user change the zoom factor by dragging a box within the zoomed data window of the transformer. This box indicates the area to be zoomed in or zoomed out.
Edit-Point Interactor	Lets the user modify a data point by dragging its graphical representation within the data display area.
Highlight-Point Interactor	Triggers an interaction event whenever the mouse moves over a data point in the data display area.
Information-View Interactor	Displays information about a data point whenever the user moves the mouse over the data point in the data display area.
Pick-Data-Points Interactor	Triggers an event when the user selects data by clicking a projected point in the data display area.

General Architecture of JViews Charts

Describes the general architecture and the main components of IBM® ILOG® JViews Charts.

In this section

Data model
Describes the data model on which JViews Charts is based.

Graphical representation
Describes the various chart renderers.

Binding data model and graphical representation
Describes how the binding between the data model and the graphical representation is performed.

The style sheet
Describes the CSS mechanism in JViews Charts.

Data model

IBM® ILOG® JViews Charts is based on the Separable Model Architecture. This architecture is a variant of the Model-View-Controller model that was introduced in Swing. In this design, the model manages the data or the values represented by the chart component, while the view manages the graphical representation of the model, and handles interactions on it.

The use of this design in JViews Charts allows you to have a clear distinction between data model that handles the sets of data by means of data sources (or data sets), and data display that draws the graphical representation of data.

The data model handles the data sets by means of data sources. The data set is a single set of data points. A data point is defined by an (X, Y) coordinate pair expressed with double primitives, and an optional label.

The contents of a data source is dynamic and its type depends on the origin of the data (XML, JDBC, Flat file or In-Memory).

The data model distinguishes data series from sources of data. Having distinct entities to represent an elementary set of data and the whole data provides the following benefits:

- Data sets exist as objects rather than internal references within the data model, which makes it easier to reference and use them in an application.
- You can create new data set types and use them with existing data sources. Likewise, you can create new data source types that handle existing data sets.
- You can easily mix data sets that come from different sources (for example, data sets extracted from a database query with data sets whose values are updated by a thread).
- You can create data sets as combinations or wrappers of other data sets.

The data model can comprise Java™ classes or an implicit default model specified in an XML file, a flat file, a database or in-memory.

If you create a new chart with the Charts Designer, you load data from an external data source (XML file, flat file, database, or in-memory Java classes) or from a template.

If you create a new chart using the API, you will use the following interfaces:

- The `IlvDataSet` interface acts as a data holder. The objects that implement this interface manage data as a set of data points, and provide the required API (Application Programming Interface) to fetch and modify data. An abstract implementation of this interface called `IlvAbstractDataSet` is provided in the library as well as several concrete implementations. See Using the Data Model in *Developing with the SDK*.
- The `IlvDataSource` interface acts as a source of data sets. It handles a collection of data sets and provides the required API to access them.The implementation of this interface allows you to import data from an external package (for example a database, an XML file, and so on) into the Charts library. An abstract implementation called `IlvAbstractDataSource` is provided in the library as a starting point for your own custom implementation. A concrete memory-based implementation is provided by the `IlvDefaultDataSource` class.

Graphical representation

The graphical representation of a data source is defined according to the following criteria:

- The global characteristics of a chart (for example, whether it uses a cartesian or polar projection).
- The conversion from data space to screen coordinates is handled by a projector object, which is an instance of `IlvChartProjector`. The type of a projector object is defined by the type of the chart, and is a global parameter of a chart:
  - A Cartesian projector when using a Cartesian chart.
  - A polar projector when using a radar, polar, or pie chart.

The way data is rendered on the screen (as a polyline, a bar, or a bubble) is handled by chart renderers, which are instances of subclasses of `IlvChartRenderer`.

There are three types of chart renderers:

- Composite renderers (instances of `IlvCompositeChartRenderer`).

  The composite renderers are used to render the contents of a data source using a collection of child chart renderers so that each data set in the associated data source is rendered by one of these child renderers.

  Depending on the rendering type, the relation between a data set and a child renderer can be one of the following:

  - unary relation: one child renderer per data set.

    The unary relations are handled by instances of subclasses of `IlvSimpleCompositeChartRenderer`. An example of a simple composite chart renderer is the polyline renderer, which renders each data set of the data source by a polyline.

  - *n*-ary relation: one child renderer using several data sets.

    The *n*-ary relations are handled by direct subclasses of `IlvCompositeChartRenderer`. An example of a composite chart renderer that uses several data sets for one graphical representation is the high-low renderer (*hilo)*, which needs two data sets (high and low values) for one graphical representation.

- Single renderers (instances of `IlvSingleChartRenderer`).

  Single renderers are used as elementary chart renderers by composite renderers to draw the graphical representation of data sets. While a composite renderer handles a data source, a single renderer handles a data set. Single renderers can also be used directly when you handle specific data set.

- Simple renderers (instances of `IlvSimpleChartRenderer`)

  Simple renderers are used to render the contents of a data source without making use of other renderers for particular data sets.

Binding data model and graphical representation

The binding between a chart and the data model is performed through a unary relationship from a chart renderer to a data source.

To associate a data source with a chart renderer:

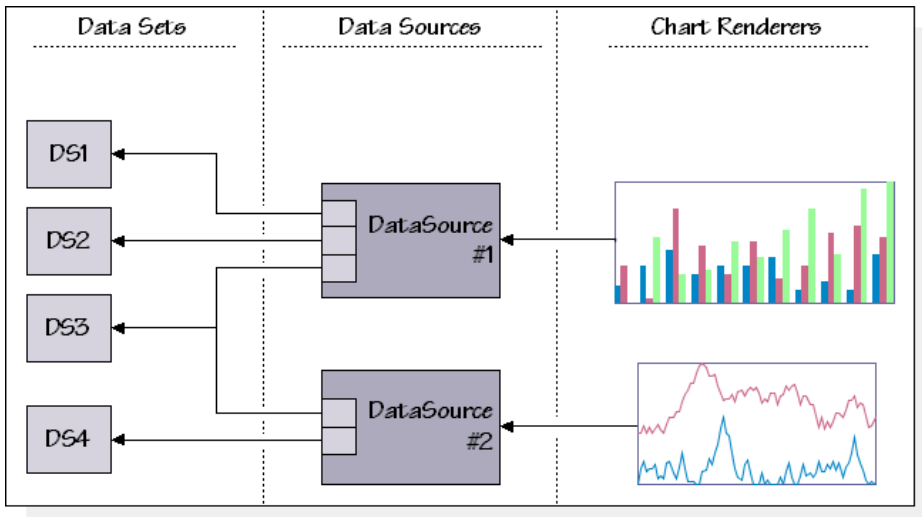
- Use the following method defined in the `IlvChartRenderer` class:

```
void setDataSource(IlvDataSource dataSource)
```

To obtain the current data source of a chart renderer:

- Use the following method:

```
IlvDataSource getDataSource
```



The style sheet

The appearance of a chart can be dynamically controlled with cascading style sheets (CSS).

Cascading style sheets (CSS) are a powerful mechanism to customize HTML rendering inside a Web browser. It comes from the W3C, and has now reached the state of a standard recommendation. Here we transpose the CSS level 2 recommendation in the Java language and use it to set Bean properties according to the Java object hierarchy and state. The CSS selector was designed to match HTML or XML documents. However, it can be used to match a hierarchy of Java objects accessible from a model interface. The declarations are then sorted for the model objects and used according to the application that controls the styling engine.

In JViews Charts, the styling engine is responsible for creating and customizing chart components and the data graphical representation at load time. At run time, the engine customizes the data graphical representation according to the model changes.

Note: For more information on Cascading style sheets, see Using CSS Syntax in the Style Sheet in Developing with the SDK.

When you create a chart, you can define your own style rules for:

- Chart Component

  The CSS rules are used to customize the global appearance of the chart. See Customizing your chart in *Using the Designer*.

- Chart Data

  The CSS rules are used to control how individual data points or data series are rendered. See Managing data style rules in *Using the Designer*.

Developing with JViews Charts

Describes the process for building a chart component with IBM® ILOG® JViews Charts.

In this section

The process flow
Describes the process through which a chart is created.

Basic steps for building a chart component
Describes the basic steps for creating a chart.

Creating a chart using the Designer
Explains how to use the Designer for JViews Charts for creating and configuring interactive charts.

When to use the API
Lists the decision points at which you may decide to use the Java API.

The process flow

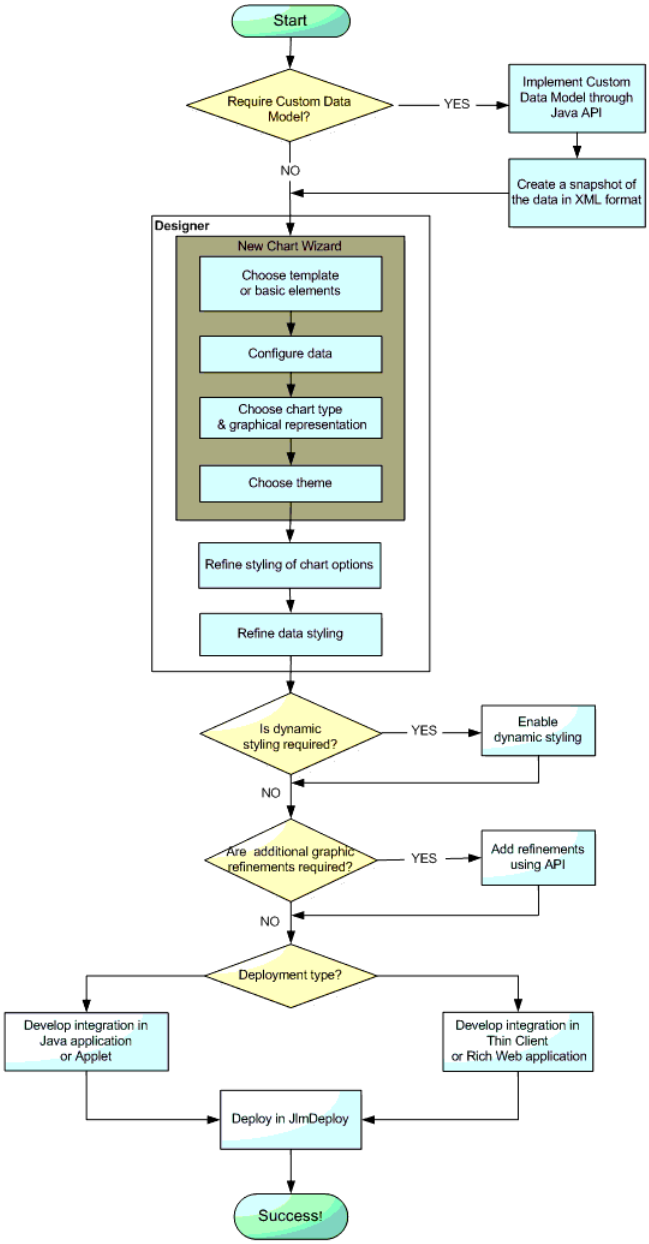
You can create your chart through the GUI of the Designer for JViews Charts and extend the development of your chart by using the SDK.

There is no reason that would prevent you from carrying out most of your development in the Designer (see *Creating a chart using the Designer*). However, at a certain point you might need to use the API to extend the development of your chart (see *When to use the API*).

Figure 4.1 shows the process flow for building a chart component in JViews Charts; it presents a high-level view of the overall process. The detailed tasks belonging to each step are described in the user documents:

- Using the Designer
- Developing the SDK

Unless otherwise specified, all the steps of the process flow can be carried out through the Designer and are documented in Using the Designer.



The Process Flow Diagram

You can develop a chart from scratch or by connecting to existing data, with the Designer. Working with the Designer, you develop components entirely through the GUI of the Designer. However, you might need to write code to do one of the following:

- add graphic refinements if you need more sophisticated features,
- integrate your component into an application,
- deploy an application as a thin client.

For more information, see *When to use the API*.

Basic steps for building a chart component

Populating the chart.

- To populate your chart you have to load data from data sources. The following predefined types of data sources are available:
  - *XML*
  - *Flat File*
  - *Database (JDBC)*
  - *In-memory*
  - *Data Writeback*
  - *Customized Data Sources*

XML

This type of data source allows you to load data from an XML file. The expected format is an application of the W3C XML language. You can find the full Document Type Definition of this format in Document Type Definition for XML data file in *Using the Designer*.

Flat File

This type of data source is a text file containing the same number of values on each line. The values are separated by a separator character. The supported separator characters are comma, semicolon, space, and tab. You can request autodetection of the separators. If a flat file contains values exported from Microsoft® Excel, it is sometimes referred to as a CSV (comma-separated values) file. In a flat file, there is an implicit data model in which the first line contains the names of the model properties and each subsequent line contains the property values for an object, with the properties in the same order as on the first line.

Database (JDBC)

This type of data source allows you to retrieve data values from database servers by using the JDBC interface. JDBC technology offers a platform and server independent way to retrieve data stored in a database. A database contains tables of columns and rows. Each row in a table represents an object in the data model. Each column in a table represents an attribute and can be mapped for use in the chart. An Excel file (`.xls`) is recognized as a database; the worksheets are treated as tables.

In-memory

The supplied in-memory data model comprises Java™ classes that conform to the JViews Charts data model, with the data belonging to the Basic template. This type of data source supports writing operations such as appending a new data point or changing values of an existing data point.

Data Writeback

The XML data source (`IlvXMLDataSource`) and the flat file data source (based on the class `IlvSwingTableDataSource`) are read-write. This means that the `IlvDataSet.setData` method can be used on them; the modified values are held in memory but not automatically written to a file . If you need modified values to be written to a file, you can implement this functionality. In the XML case, the class `IlvXMLDataWriter` is useful for this purpose.

The JDBC data source (class `IlvJDBCDataSource`) operates in a similar way when created in read-only mode. When created in read-write mode, it writes back modifications to the database.

Customized Data Sources

If the data you want to display is not of one of these predefined types, you can extend the data model to create a customized data source. To see how to do this, see the section Extending the Data Model in *Developing with the SDK*.

Before you move on to the next step, that is, styling your chart using the Designer, you need to create a snapshot of the data source in XML format. This JViews Charts XML file serves as input to the Designer. To create this file, you can use the `IlvXMLDataWriter` class, as explained in Reading and writing data from an XML source in *Developing with the SDK*.

Styling your chart.

The representation in your chart of the objects that form the business data, including the way your end users interact with them, can be customized by applying Cascading Style Sheets (CSS).

1. You can change the representation of the data, by adjusting the styling properties of the objects, affecting the following features:

   - The color and the outline of the graphical reprensentation of the series.
   - The shapes of the data points marker.
   - The graduation of the scale.
   - The appearance of the grid lines.
   - The position of the legend.

   See Customizing your chart in *Using the Designer*.

2. You can also apply styling conditions by creating and editing rules. This capability is particularly useful if you need to style data dynamically as the model changes. JViews Charts provides an easy-to-use natural language editor to help you write style rules without having to know the detailed CSS syntax. It allows you to develop conditions for applying specific styling features. See Managing data style rules in *Using the Designer*.

3. The result of the work you have done within the Designer is a project file with the extension `.icpr` (JViews Chart Project) and a style sheet with the extension `.css` (Cascading Style Sheet).

   If the data to be displayed comes from one of the predefined data source types, you can integrate the styling and the data source together into the chart, by using the method `setProject(java.net.URL)`.

   If the data to be displayed comes form a custom data source, you need to integrate the styling into the chart by using the method `setStyleSheet(java.lang.String)`.

Adding interaction.

JViews Charts provides interactors that allow the user to interact with a chart. Chart interactors let you associate one or several behaviors to a chart object; they define atomic

interactions that can be combined together and extended to achieve complex interactive functionalities.

- To add interactions, see the following sections:
  - Interacting With Charts in *Developing with the S DK* for a Java application or an applet.
  - Installing interactors in a chart in *Building Web Applications* for a thin-client JSF DHTML.
  - Set up interactions in *Building Web Applications* for a Rich Web Client application.

Integrating your chart into an application.

Now that you have created your JViews chart, you need to integrate it into an application.

- To lintegrate your chart component into an application see Integrating your development into an application and Writing an Application in *Using the Designer*.

  You can deploy the application in different ways:

  - Swing GUI, applet, DHTML thin client. See Deploying an application as a DHTML-only thin client in *Building Web Applications*).
  - DHMTL-based JSF thin client. See Using DHTML-based JSF components to build Web applications in *Building Web Applications*.
  - Rich Web Client. See Creating Rich Web Charts in *Building Web Applications*.

  Essentially, to integrate a chart component into a GUI, you load a project. The project construct groups the data source and style sheet which are the basis for a chart.

  Deployment as an applet is the same as deployment as an application; it just requires a line of HTML code for the applet in addition to the application code.

  Deployment as a thin client is facilitated by the servlet support in the Java API, and by the JSF tag library for the thin client (see The JViews Charts Faces component set in *Building Web Applications*).

  Deployment as a Rich Web Client is facilitated by the JSF tag library for the Rich Web Client (see The tag library in *Building Web Applications*).

Creating a chart using the Designer

The main advantage is its simplicity and usability: in a few mouse clicks you will create and customize your charts through an intuitive and easy-to-use interface.

Note: Treemap charts are currently not available.

Creating a new chart.

1. Use the New Chart Wizard, which is open by default when you launch the Designer. The New Chart Wizard allows you to create a chart based on a template, or by loading data from a data source. (See Creating a new basic chart in *Using the Designer*.)
2. You can create your charts by using JViews Charts templates. JViews Charts template determines the basic structure for a chart and contains settings such as specific data model and style. You can also create your own templates from the current styling and data model. (See Using templates in *Using the Designer*.)
3. You can also choose among a predefined set of data sources:
   - Flat file: loads data from an existing text file (comma-separated or tab-separated values). See Loading data from a flat file in *Using the Designer*.
   - XML file: loads data from an XML file. See Loading your data in *Using the Designer*.
   - JDBC database: retrieves data values from database servers. See Loading data from database (JDBC) in *Using the Designer*.
   - In-memory: data points are stored in memory with arrays of double primitives. See Loading data from In-Memory in *Using the Designer*.

Customizing a chart.

- You can customize your charts in different ways to improve their appearance. (See Using More Designer Features in *Using the Designer*.)

Testing a chart.

- Switch to the Preview Mode to see the behavior of the chart as it will be in your application. This is useful for testing the appearances and behavior implemented through styling. (See Testing application behavior in *Using the Designer*.)

Saving your work as a project.

- Save your chart with the extension `.icpr` (JViews Charts Project).

  A project file is an association of a data source type (flat file, XML, JDBC or in-memory) and a style sheet (`.css` file) and it is created when you save your chart for the first time.

When you save your work within the Designer, you saves three files: a project file, a style sheet, and a data file. The project file specifies the name of the style sheet file, the type and URL of the data source, and the name of the data file.

When to use the API

The decision points at which you may decide to use the Java API are the following:

- Extend the Data Model.

  If your data cannot be connected by JDBC or is not in XML or is not held in an in-memory data model, you must use the Java API to integrate your data into the Designer before starting development of your chart component. See Extending the Data Model in *Developing with the SDK*.

- Extend the graphic refinement of your chart component.

  If after you complete a development cycle in the Designer you find that you require graphic refinement features that are not available in the GUI, you can achieve these by using the Java API directly. See Writing a new grid in *Developing with the SDK*.

  This is not applicable for developing Rich Web Client applications.

- Write your own interactor.

  See Writing your own interactor in *Developing with the SDK*.

  This is not applicable for developing thin client and Rich Web Client applications.

- Integrate your component into an application.

  See Integrating your development into an application in *Using the Designer*.

- Print.

  See Printing in *Developing with the SDK*.

- Deploy an application as a thin client.

  See Deploying an application as a DHTML-only thin client in *Building Web Applications*.

- Deploy an application as a Rich Web Client.

  See Creating Rich Web Charts in *Building Web Applications*.

Index