NAME

barchart − Bar chart for plotting X-Y coordinate data.

SYNOPSIS

barchart *pathName* ?*option value*?...

DESCRIPTION

The **barchart** command creates a bar chart for plotting two-dimensional data (X-Y coordinates). A bar chart is a graphic means of comparing numbers by displaying bars of lengths proportional to the y-coordinates of the points they represented. The bar chart has many configurable components: coordinate axes, elements, legend, grid lines, cross hairs, etc. They allow you to customize the look and feel of the graph.

INTRODUCTION

The **barchart** command creates a new window for plotting two-dimensional data (X-Y coordinates), using bars of various lengths to represent the data points. The bars are drawn in a rectangular area displayed in the center of the new window. This is the*plotting ar ea*. The coordinate axes are drawn in the margins surrounding the plotting area. By default, the legend is drawn in the right margin. The title is displayed in top margin.

A **barchart** widget has several configurable components: coordinate axes, data elements, legend, grid, cross hairs, pens, postscript, and annotation markers. Each component can be queried or modified.

axis

Up to four coordinate axes (two X−coordinate and two Y−coordinate axes) can be displayed, but you can create and use any number of axes. Axes control what region of data is displayed and how the data is scaled. Each axis consists of the axis line, title, major and minor ticks, and tick labels. Tick labels display the value at each major tick.

crosshairs Cross hairs are used to position the mouse pointer relative to the X and Y coordinate axes. Two perpendicular lines, intersecting at the current location of the mouse, extend across the plotting area to the coordinate axes.

element An element represents a set of data to be plotted. It contains an x and y vector of values representing the data points. Each data point is displayed as a bar where the length of the bar is proportional to the ordinate (Y-coordinate) of the data point. The appearance of the bar, such as its color, stipple, or relief is configurable.

A special case exists when two or more data points have the same abscissa (X-coordinate). By default, the bars are overlayed, one on top of the other. The bars are drawn in the order of the element display list. But you can also configure the bars to be displayed in two other ways. They may be displayed as a stack, where each bar (with the same abscissa) is stacked on the previous. Or they can be drawn side-by-side as thin bars. The width of each bar is a function of the number of data points with the same abscissa.

grid Extends the major and minor ticks of the X−axis and/or Y−axis across the plotting area.

legend The legend displays the name and symbol of each data element. The legend can be drawn in any margin or in the plotting area.

marker Markers are used annotate or highlight areas of the graph. For example, you could use a text marker to label a particular data point. Markers come in various forms: text strings, bitmaps, connected line segments, images, polygons, or embedded widgets.

pen Pens define attributes for elements. Data elements use pens to specify how they should be drawn. A data element may use many pens at once. Here the particular pen used for a data point is determined from each element's weight vector (see the element's **−weight** and **−style** options).

postscript The widget can generate encapsulated PostScript output. This component has several options to configure how the PostScript is generated.

SYNTAX

barchart *pathName* ?*option value*?... The**bar chart** command creates a new window *pathName* and makes it into a **barchart** widget. At the time this command is invoked, there must not exist a window named *pathName*, but *pathName*'s parent must exist. Additional options may be specified on the command line or in the option database to configure aspects of the graph such as its colors and font. See the **configure** operation below for the exact details about what *option* and *value* pairs are valid.

If successful, **barchart** returns the path name of the widget. It also creates a new Tcl command by the same name. You can use this command to invoke various operations that query or modify the graph. The general form is: *pathName operation* ?*arg*?... Both*oper ation* and its arguments determine the exact behavior of the command. The operations available for the graph are described in the **BARCHART OPERATIONS** section.

The command can also be used to access components of the graph. *pathName component oper ation* ?*arg*?... The operation, now located after the name of the component, is the function to be performed on that component. Each component has its own set of operations that manipulate that component. They will be described below in their own sections.

EXAMPLE

The **barchart** command creates a new bar chart.

```
# Create a new bar chart.  Plotting area is black.
barchart .b -plotbackground black
```

A new Tcl command **.b** is created. This command can be used to query and modify the bar chart. For example, to change the title of the graph to "My Plot", you use the new command and the **configure** operation.

```
# Change the title.
.b configure -title "My Plot"
```

To add data elements, you use the command and the **element** component.

```
# Create a new element named "e1"
.b element create e1 \
   -xdata { 1 2 3 4 5 6 7 8 9 10 } \
   -ydata { 26.18 50.46 72.85 93.31 111.86 128.47 143.14
      155.85 166.60 175.38 }
```

The element's X-Y coordinates are specified using lists of numbers. Alternately, BLT vectors could be used to hold the X-Y coordinates.

```
# Create two vectors and add them to the barchart.
vector xVector yVector
xVector set { 1 2 3 4 5 6 7 8 9 10 }
yVector set { 26.18 50.46 72.85 93.31 111.86 128.47 143.14 155.85
   166.60 175.38 }
n.b element create e1 -xdata xVector -ydata yVector
```

The advantage of using vectors is that when you modify one, the graph is automatically redrawn to reflect the new values.

```
# Change the y coordinate of the first point.
set yVector(0) 25.18
```

An element named **e1** is now created in **.b**. It is automatically added to the display list of elements. You can use this list to control in what order elements are displayed. To query or reset the element display list, you use the element's **show** operation.

```
# Get the current display list
set elemList [.b element show]
# Remove the first element so it won't be displayed.
.b element show [lrange $elemList 0 end]
```

The element will be displayed by as many bars as there are data points (in this case there are ten). The bars will be drawn centered at the x-coordinate of the data point. All the bars will have the same attributes (colors, stipple, etc). The width of each bar is by default one unit. You can change this with using the **−barwidth** option.

```
# Change the scale of the x-coordinate data
xVector set { 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0 }
# Make sure we change the bar width too.
.b configure -barwidth 0.2
```

The height of each bar is proportional to the ordinate (Y-coordinate) of the data point.

If two or more data points have the same abscissa (X-coordinate value), the bars representing those data points may be drawn in various ways. The default is to overlay the bars, one on top of the other. The ordering is determined from the of element display list. If the stacked mode is selected (using the **−barmode** configuration option), the bars are stacked, each bar above the previous.

```
# Display the elements as stacked.
.b configure -barmode stacked
```

If the aligned mode is selected, the bars having the same x-coordinates are displayed side by side. The width of each bar is a fraction of its normal width, based upon the number of bars with the same x-coordinate.

```
# Display the elements side-by-side.
.b configure -barmode aligned
```

By default, the element's label in the legend will be also **e1**. You can change the label, or specify no legend entry, again using the element's **configure** operation.

```
# Don't display "e1" in the legend.
.b element configure e1 -label ""
```

You can configure more than just the element's label. An element has many attributes such as stipple, foreground and background colors, relief, etc.

```
.b element configure e1 -fg red -bg pink \
   -stipple gray50
```

Four coordinate axes are automatically created: **x**, **x2**, **y**, and **y2**. And by default, elements are mapped onto the axes **x** and **y**. This can be changed with the**−mapx** and **−mapy** options.

```
# Map "e1" on the alternate y axis "y2".
.b element configure e1 -mapy y2
```

Axes can be configured in many ways too. For example, you change the scale of the Y−axis from linear to log using the **axis** component.

```
# Y-axis is log scale.
.b axis configure y -logscale yes
```

One important way axes are used is to zoom in on a particular data region. Zooming is done by simply specifying new axis limits using the **−min** and **−max** configuration options.

```
.b axis configure x −min 1.0 −max 1.5
.b axis configure y −min 12.0 −max 55.15
```

To zoom interactively, you link the**axis configure** operations with some user interaction (such as pressing the mouse button), using the **bind** command. To convert between screen and graph coordinates, use the **invtransform** operation.

```
# Click the button to set a new minimum
bind .b <ButtonPress-1> {
    %W axis configure x −min [%W axis invtransform x %x]
    %W axis configure x −min [%W axis invtransform x %y]
}
```

By default, the limits of the axis are determined from data values. To reset back to the default limits, set the **−min** and **−max** options to the empty value.

```
# Reset the axes to autoscale again.
.b axis configure x −min {} −max {}
.b axis configure y −min {} −max {}
```

By default, the legend is drawn in the right margin. You can change this or any legend configuration options using the **legend** component.

```
# Configure the legend font, color, and relief
.b legend configure -position left -relief raised \
   -font fixed -fg blue
```

To prevent the legend from being displayed, turn on the **−hide** option.

```
# Don’t display the legend.
.b legend configure −hide yes
```

The **barchart** has simple drawing procedures called markers. They can be used to highlight or annotate data in the graph. The types of markers available are bitmaps, polygons, lines, or windows. Markers can be used, for example, to mark or brush points. For example there may be a line marker which indicates some low-water value. Markers are created using the **marker** operation.

```
# Create a line represent the low water mark at 10.0
.b marker create line -name "low_water" \
   -coords { -Inf 10.0 Inf 10.0 } \
   -dashes { 2 4 2 } -fg red -bg blue
```

This creates a line marker named **low_water**. It will display a horizontal line stretching across the plotting area at the y-coordinate 10.0. The coordinates "-Inf" and "Inf" indicate the relative minimum and maximum of the axis (in this case the x-axis). By default, markers are drawn last, on top of the bars. You can change this with the **−under** option.

```
# Draw the marker before elements are drawn.
.b marker configure low_water -under yes
```

You can add cross hairs or grid lines using the **crosshairs** and **grid** components.

```
# Display both cross hairs and grid lines.
.b crosshairs configure -hide no -color red
.b grid configure -hide no -dashes { 2 2 }
```

Finally, to get hardcopy of the graph, use the **postscript** component.

```
# Print the bar chart into file "file.ps"
.b postscript output file.ps -maxpect yes -decorations no
```

This generates a file **file.ps** containing the encapsulated PostScript of the graph. The option **−maxpect** says to scale the plot to the size of the page. Turning off the **−decorations** option denotes that no borders or color backgrounds should be drawn (i.e. the background of the margins, legend, and plotting area will be white).

SYNTAX

barchart *pathName* ?*option value*?... The**bar chart** command creates a new window *pathName* and makes it into a barchart widget. At the time this command is invoked, there must not exist a window named *pathName*, but *pathName*'s parent must exist. Additional options may may be specified on the command line or in the option database to configure aspects of the bar chart such as its colors and font. See the **configure** operation below for the exact details as to what *option* and *value* pairs are valid.

If successful, **barchart** returns *pathName*. It also creates a new Tcl command *pathName*. This command may be used to invoke various operations to query or modify the bar chart. It has the general form: *pathName operation* ?*arg*?... Both*oper ation* and its arguments determine the exact behavior of the command. The operations available for the bar chart are described in the following section.

BARCHART OPERATIONS

pathName **bar** *elemName* ?*option value*?...

Creates a new barchart element *elemName*. It's an error if an element *elemName* already exists. See the manual for **barchart** for details about what *option* and *value* pairs are valid.

pathName **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may be any option described below for the **configure** operation.

pathName **configure** ?*option value*?...

Queries or modifies the configuration options of the graph. If *option* isn't specified, a list describing the current options for *pathName* is returned. If *option* is specified, but not *value*, then a list describing *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the option *option* is set to *value*. The following options are valid.

−background *color*

Sets the background color. This includes the margins and legend, but not the plotting area.

−barmode *mode*
Indicates how related bar elements will be drawn. Related elements have data points with the same abscissas (X-coordinates). *Mode* indicates how those segments should be drawn. *Mode* can be **infront**, **aligned**, **overlap**, or **stacked**. The default mode is **infront**.

infront	Each successive segment is drawn in front of the previous.
stacked	Each successive segment is stacked vertically on top of the previous.
aligned	Segments is displayed aligned from right-to-left.
overlap	Like **aligned** but segments slightly overlap each other.

−barwidth *value*
Specifies the width of the bars. This value can be overrided by the individual elements using their **−barwidth** configuration option. *Value* is the width in terms of graph coordinates. The default width is **1.0**.

−borderwidth *pixels*
Sets the width of the 3−D border around the outside edge of the widget. The **−relief** option determines if the border is to be drawn. The default is **2**.

−bottommargin *pixels*
Specifies the size of the margin below the X−coordinate axis. If *pixels* is **0**, the size of the margin is selected automatically. The default is **0**.

−bufferelements *boolean*
Indicates whether an internal pixmap to buffer the display of data elements should be used. If*boolean* is true, data elements are dra wn to an internal pixmap. This option is especially useful when the graph is redrawn frequently while the remains data unchanged (for example, moving a marker across the plot). See the **SPEED TIPS** section. The default is **1**.

−cursor *cursor*
Specifies the widget's cursor. The default cursor is **crosshair**.

−font *fontName*
Specifies the font of the graph title. The default is ***-Helvetica-Bold-R-Normal-*-18-180-***.

−halo *pixels*
Specifies a maximum distance to consider when searching for the closest data point (see the element's **closest** operation below). Data points further than*pixels* a way are ignored. The default is **0.5i**.

−height *pixels*
Specifies the requested height of widget. The default is **4i**.

−invertxy *boolean*
Indicates whether the placement X−axis and Y−axis should be inverted. If*boolean* is true, the X and Y axes are swapped. The default is **0**.

−justify *justify*
Specifies how the title should be justified. This matters only when the title contains more than one line of text. *Justify* must be **left**, **right**, or **center**. The default is **center**.

−leftmargin *pixels*
Sets the size of the margin from the left edge of the window to the Y−coordinate axis. If *pixels* is **0**, the size is calculated automatically. The default is **0**.

−plotbackground *color*
Specifies the background color of the plotting area. The default is **white**.

−plotborderwidth *pixels*
Sets the width of the 3-D border around the plotting area. The **−plotrelief** option determines if a border is drawn. The default is **2**.

−plotpadx *pad*
Sets the amount of padding to be added to the left and right sides of the plotting area. *Pad* can be a list of one or two screen distances. If *pad* has two elements, the left side of the plotting area entry is padded by the first distance and the right side by the second. If *pad* is just one distance, both the left and right sides are padded evenly. The default is **8**.

−plotpady *pad*
Sets the amount of padding to be added to the top and bottom of the plotting area. *P ad* can be a list of one or two screen distances. If *pad* has two elements, the top of the plotting area is padded by the first distance and the bottom by the second. If *pad* is just one distance, both the top and bottom are padded evenly. The default is **8**.

−plotrelief *relief*
Specifies the 3-D effect for the plotting area. *Relief* specifies ho w the interior of the plotting area should appear relative to rest of the graph; for example, **raised** means the plot should appear to protrude from the graph, relative to the surface of the graph. The default is **sunken**.

−relief *relief*
Specifies the 3-D effect for the barchart widget. *Relief* specifies ho w the graph should appear relative to widget it is packed into; for example, **raised** means the graph should appear to protrude. The default is **flat**.

−rightmargin *pixels*
Sets the size of margin from the plotting area to the right edge of the window. By default, the legend is drawn in this margin. If*pixels* is than 1, the mar gin size is selected automatically.

−takefocus *focus*
Provides information used when moving the focus from window to window via keyboard traversal (e.g., Tab and Shift-Tab). If*focus* is **0**, this means that this windo w should be skipped entirely during keyboard traversal. **1** means that the this window should always receive the input focus. An empty value means that the traversal scripts make the decision whether to focus on the window. The default is "".

−tile *image*
Specifies a tiled background for the widget. If *image* isn't "", the background is tiled using *image*. Otherwise, the normal background color is drawn (see the **−background** option). *Image* must be an image created using the Tk **image** command. The default is "".

−title *text*
Sets the title to *text*. If *text* is "", no title will be displayed.

−topmargin *pixels*
Specifies the size of the margin above the x2 axis. If *pixels* is **0**, the margin size is calculated automatically.

−width *pixels*
Specifies the requested width of the widget. The default is **5i**.

pathName **crosshairs** *operation* ?*arg*?
See the **CROSSHAIRS COMPONENT** section.

pathName **element** *operation* ?*arg*?...
See the **ELEMENT COMPONENTS** section.

pathName **extents** *item*
Returns the size of a particular item in the graph. *Item* must be either **leftmar gin**, **rightmargin**, **topmargin**, **bottommargin**, **plotwidth**, or **plotheight**.

pathName **grid** *operation* ?*arg*?...
See the **GRID COMPONENT** section.

pathName **invtransform** *winX winY*
Performs an inverse coordinate transformation, mapping window coordinates back to graph coordinates, using the standard X−axis and Y−axis. Returns a list of containing the X-Y graph coordinates.

pathName **inside** *x y*
Returns **1** is the designated screen coordinate (*x* and *y*) is inside the plotting area and **0** otherwise.

pathName **legend** *operation* ?*arg*?...
See the **LEGEND COMPONENT** section.

pathName **line operation arg**...
The operation is the same as **element**.

pathName **marker** *operation* ?*arg*?...
See the **MARKER COMPONENTS** section.

pathName **metafile** ?*fileName*?
This operation is for Window platforms only. Creates a Windows enhanced metafile of the barchart. If present,*fileName* is the file name of the ne w metafile. Otherwise, the metafile is automatically added to the clipboard.

pathName **postscript** *operation* ?*arg*?...
See the **POSTSCRIPT COMPONENT** section.

pathName **snap** *photoName*
Takes a snapshot of the graph and stores the contents in the photo image *photoName*. *PhotoName* is the name of a Tk photo image that must already exist.

pathName **transform** *x y*
Performs a coordinate transformation, mapping graph coordinates to window coordinates, using the standard X−axis and Y−axis. Returns a list containing the X−Y screen coordinates.

pathName **xaxis** *operation* ?*arg*?...

pathName **x2axis** *operation* ?*arg*?...

pathName **yaxis** *operation* ?*arg*?...

pathName **y2axis** *operation* ?*arg*?...
See the **AXIS COMPONENTS** section.

BARCHART COMPONENTS

A graph is composed of several components: coordinate axes, data elements, legend, grid, cross hairs, postscript, and annotation markers. Instead of one big set of configuration options and operations, the graph is partitioned, where each component has its own configuration options and operations that specifically control that aspect or part of the graph.

AXIS COMPONENTS

Four coordinate axes are automatically created: two X−coordinate axes (**x** and **x2**) and two Y−coordinate axes (**y**, and **y2**). By default, the axis **x** is located in the bottom margin, **y** in the left margin, **x2** in the top margin, and **y2** in the right margin.

An axis consists of the axis line, title, major and minor ticks, and tick labels. Major ticks are drawn at uniform intervals along the axis. Each tick is labeled with its coordinate value. Minor ticks are drawn at uniform intervals within major ticks.

The range of the axis controls what region of data is plotted. Data points outside the minimum and maximum limits of the axis are not plotted. By default, the minimum and maximum limits are determined from the data, but you can reset either limit.

You can create and use several axes. To create an axis, invoke the axis component and its create operation.

```
# Create a new axis called "temperature"
.b axis create temperature
```

You map data elements to an axis using the element's −mapy and −mapx configuration options. They specify the coordinate axes an element is mapped onto.

```
# Now map the temperature data to this axis.
.b element create "temp" −xdata $x −ydata $tempData \
    −mapy temperature
```

While you can have many axes, only four axes can be displayed simultaneously. They are drawn in each of the margins surrounding the plotting area. The axes **x** and **y** are drawn in the bottom and left margins. The axes **x2** and **y2** are drawn in top and right margins. Only**x** and **y** are sho wn by default. Note that the axes can have different scales.

To display a different axis, you invoke one of the following components: **xaxis**, **yaxis**, **x2axis**, and **y2axis**. The **use** operation designates the axis to be drawn in the corresponding margin: **xaxis** in the bottom, **yaxis** in the left, **x2axis** in the top, and **y2axis** in the right.

```
# Display the axis temperature in the left margin.
.b yaxis use temperature
```

You can configure axes in many ways. The axis scale can be linear or logarithmic. The values along the axis can either monotonically increase or decrease. If you need custom tick labels, you can specify a Tcl procedure to format the label any way you wish. You can control how ticks are drawn, by changing the major tick interval or the number of minor ticks. You can define non-uniform tick intervals, such as for time-series plots.

pathName **axis cget** *axisName option*

Returns the current value of the option given by *option* for *axisName*. *Option* may be any option described below for the axis **configure** operation.

pathName **axis configure** *axisName* ?*axisName*?... ?*option value*?...

Queries or modifies the configuration options of *axisName*. Several axes can be changed. If *option* isn't specified, a list describing all the current options for *axisName* is returned. If *option* is specified, but not *value*, then a list describing *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the axis option *option* is set to *value*. The following options are valid for axes.

−autorange *range*

Sets the range of values for the axis to *range*. The axis limits are automatically reset to display the most recent data points in this range. If *range* is 0.0, the range is determined from the limits of the data. If **−min** or **-max** are specified, they override this option. The default is **0.0**.

−color *color*
Sets the color of the axis and tick labels. The default is **black**.

−command *prefix*
Specifies a Tcl command to be invoked when formatting the axis tick labels. *Prefix* is a string containing the name of a Tcl proc and any extra arguments for the procedure. This command is invoked for each major tick on the axis. Two additional arguments are passed to the procedure: the pathname of the widget and the current the numeric value of the tick. The procedure returns the formatted tick label. If "" is returned, no label will appear next to the tick. You can get the standard tick labels again by setting *prefix* to "". The default is "".

Please note that this procedure is invoked while the bar chart is redrawn. You may query the widget's configuration options. But do not reset options, because this can have unexpected results.

−descending *boolean*
Indicates whether the values along the axis are monotonically increasing or decreasing. If *boolean* is true, the axis values will be decreasing. The default is **0**.

−hide *boolean*
Indicates whether the axis is displayed.

−justify *justify*
Specifies how the axis title should be justified. This matters only when the axis title contains more than one line of text. *Justify* must be **left**, **right**, or **center**. The default is **center**.

−limits *formatStr*
Specifies a printf-like description to format the minimum and maximum limits of the axis. The limits are displayed at the top/bottom or left/right sides of the plotting area. *FormatStr* is a list of one or two format descriptions. If one description is supplied, both the minimum and maximum limits are formatted in the same way. If two, the first designates the format for the minimum limit, the second for the maximum. If "" is given as either description, then the that limit will not be displayed. The default is "".

−linewidth *pixels*
Sets the width of the axis and tick lines. The default is **1** pixel.

−logscale *boolean*
Indicates whether the scale of the axis is logarithmic or linear. If*boolean* is true, the axis is logarithmic. The default scale is linear.

−loose *boolean*
Indicates whether the limits of the axis should fit the data points tightly, at the outermost data points, or loosely, at the outer tick intervals. This is relevant only when the axis limit is automatically calculated. If *boolean* is true, the axis range is "loose". The default is **0**.

−majorticks *majorList*
Specifies where to display major axis ticks. You can use this option to display ticks at non-uniform intervals. *MajorList* is a list of axis coordinates designating the location of major ticks. No minor ticks are drawn. If*majorList* is "", major ticks will be automatically computed. The default is "".

−max *value*
Sets the maximum limit of *axisName*. Any data point greater than *value* is not displayed. If *value* is "", the maximum limit is calculated using the largest data value. The default is "".

−min *value*
Sets the minimum limit of *axisName*. Any data point less than *value* is not displayed. If *value* is "", the minimum limit is calculated using the smallest data value. The default is "".

−minorticks *minorList*
Specifies where to display minor axis ticks. You can use this option to display minor ticks at non-uniform intervals. *MinorList* is a list of real values, ranging from 0.0 to 1.0, designating the placement of a minor tick. No minor ticks are drawn if the **−majortick** option is also set. If *minorList* is "", minor ticks will be automatically computed. The default is "".

−rotate *theta*
Specifies the how many degrees to rotate the axis tick labels. *Theta* is a real v alue representing the number of degrees to rotate the tick labels. The default is **0.0** degrees.

−shiftby *value*
Specifies how much to automatically shift the range of the axis. When the new data exceeds the current axis maximum, the maximum is increased in increments of *value*. You can use this option to prevent the axis limits from being recomputed at each new time point. If *value* is 0.0, then no automatic shifting is down. The default is **0.0**.

−showticks *boolean*
Indicates whether axis ticks should be drawn. If *boolean* is true, ticks are drawn. If false, only the axis line is drawn. The default is **1**.

−stepsize *value*
Specifies the interval between major axis ticks. If *value* isn't a valid interval (must be less than the axis range), the request is ignored and the step size is automatically calculated.

−subdivisions *number*
Indicates how many minor axis ticks are to be drawn. For example, if *number* is two, only one minor tick is drawn. If*number* is one, no minor ticks are displayed. The def ault is **2**.

−tickfont *fontName*
Specifies the font for axis tick labels. The default is ***-Courier-Bold-R-Normal-*-100-***.

−ticklength *pixels*
Sets the length of major and minor ticks (minor ticks are half the length of major ticks). If *pixels* is less than zero, the axis will be inverted with ticks drawn pointing towards the plot. The default is **0.1i**.

−title *text*
Sets the title of the axis. If *text* is "", no axis title will be displayed.

−titlecolor *color*
Sets the color of the axis title. The default is **black**.

−titlefont *fontName*
Specifies the font for axis title. The default is ***-Helvetica-Bold-R-Normal-*-14-140-***.

Axis configuration options may be also be set by the**option** command. The resource class is **Axis**. The resource names are the names of the axes (such as **x** or **x2**).

```
option add *Barchart.Axis.Color  blue
option add *Barchart.x.LogScale  true
option add *Barchart.x2.LogScale false
```

pathName **axis create** *axisName* ?*option value*?...
Creates a new axis by the name *axisName*. No axis by the same name can already exist. *Option* and *value* are described in above in the axis **configure** operation.

pathName **axis delete** ?*axisName*?...
Deletes the named axes. An axis is not really deleted until it is not longer in use, so it's safe to delete axes mapped to elements.

pathName **axis invtransform** *axisName value*
Performs the inverse transformation, changing the screen coordinate *value* to a graph coordinate, mapping the value mapped to *axisName*. Returns the graph coordinate.

pathName **axis limits** *axisName*
Returns a list of the minimum and maximum limits for *axisName*. The order of the list is**min max**.

pathName **axis names** ?*pattern*?...
Returns a list of axes matching zero or more patterns. If no *pattern* argument is give, the names of all axes are returned.

pathName **axis transform** *axisName value*
Transforms the coordinate *value* to a screen coordinate by mapping the it to *axisName*. Returns the transformed screen coordinate.

Only four axes can be displayed simultaneously. By default, they are **x**, **y**, **x2**, and **y2**. You can swap in a different axis with **use** operation of the special axis components: **xaxis**, **x2axis**, **yaxis**, and **y2axis**.

```
.g create axis temp
.g create axis time
...
.g xaxis use temp
.g yaxis use time
```

Only the axes specified for use are displayed on the screen.

The **xaxis**, **x2axis**, **yaxis**, and **y2axis** components operate on an axis location rather than a specific axis like the more general **axis** component does. The **xaxis** component manages the X-axis located in the bottom margin (whatever axis that happens to be). Likewise, **yaxis** uses the Y-axis in the left margin, **x2axis** the top X-axis, and **y2axis** the right Y-axis.

They implicitly control the axis that is currently using to that location. By default, **xaxis** uses the **x** axis, **yaxis** uses **y**, **x2axis** uses **x2**, and **y2axis** uses **y2**. These components can be more convenient to use than always determining what axes are current being displayed by the graph.

The following operations are available for axes. They mirror exactly the operations of the **axis** component. The *axis* argument must be **xaxis**, **x2axis**, **yaxis**, or **y2axis**.

pathName axis **cget** *option*

pathName axis **configure** ?*option value*?...

pathName axis **invtransform** *value*

pathName axis **limits**

pathName axis **transform** *value*

pathName axis **use** ?*axisName*?
Designates the axis *axisName* is to be displayed at this location. *AxisName* can not be already in use at another location. This command returns the name of the axis currently using this location.

CROSSHAIRS COMPONENT

Cross hairs consist of two intersecting lines (one vertical and one horizontal) drawn completely across the plotting area. They are used to position the mouse in relation to the coordinate axes. Cross hairs differ from line markers in that they are implemented using XOR drawing primitives. This means that they can be quickly drawn and erased without redrawing the entire widget.

The following operations are available for cross hairs:

pathName **crosshairs cget** *option*

Returns the current value of the cross hairs configuration option given by *option*. *Option* may be any option described below for the cross hairs **configure** operation.

pathName **crosshairs configure** ?*option value*?...

Queries or modifies the configuration options of the cross hairs. If *option* isn't specified, a list describing all the current options for the cross hairs is returned. If *option* is specified, but not *value*, then a list describing *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the cross hairs option *option* is set to *value*. The following options are available for cross hairs.

−color *color*

Sets the color of the cross hairs. The default is **black**.

−dashes *dashList*

Sets the dash style of the cross hairs. *DashList* is a list of up to 11 numbers that alternately represent the lengths of the dashes and gaps on the cross hair lines. Each number must be between 1 and 255. If *dashList* is `""`, the cross hairs will be solid lines.

−hide *boolean*

Indicates whether cross hairs are drawn. If *boolean* is true, cross hairs are not drawn. The default is **yes**.

−linewidth *pixels*

Set the width of the cross hair lines. The default is **1**.

−position *pos*

Specifies the screen position where the cross hairs intersect. *Pos* must be in the form "*@x,y*", where *x* and *y* are the window coordinates of the intersection.

Cross hairs configuration options may be also be set by the **option** command. The resource name and class are **crosshairs** and **Crosshairs** respectively.

```
option add *Barchart.Crosshairs.LineWidth 2
option add *Barchart.Crosshairs.Color     red
```

pathName **crosshairs off**

Turns off the cross hairs.

pathName **crosshairs on**

Turns on the display of the cross hairs.

pathName **crosshairs toggle**

Toggles the current state of the cross hairs, alternately mapping and unmapping the cross hairs.

ELEMENTS

A data element represents a set of data. It contains x and y vectors which are the coordinates of the data points. Elements are displayed as bars where the length of the bar is proportional to the ordinate of the data point. Elements also control the appearance of the data, such as the color, stipple, relief, etc.

When new data elements are created, they are automatically added to a list of displayed elements. The display list controls what elements are drawn and in what order.

The following operations are available for elements.

pathName **element activate** *elemName* ?*index*?...
Specifies the data points of element *elemName* to be drawn using active foreground and background colors. *ElemName* is the name of the element and *inde x* is a number representing the index of the data point. If no indices are present then all data points become active.

pathName **element bind** *tagName* ?*sequence*? ?*command*?
Associates *command* with *tagName* such that whenever the event sequence given by *sequence* occurs for an element with this tag, *command* will be invoked. The syntax is similar to the**bind** command except that it operates on graph elements, rather than widgets. See the **bind** manual entry for complete details on *sequence* and the substitutions performed on *command* before invoking it.

If all arguments are specified then a new binding is created, replacing any existing binding for the same *sequence* and *tagName*. If the first character of*command* is **+** then *command* augments an existing binding rather than replacing it. If no *command* argument is provided then the command currently associated with *tagName* and *sequence* (it's an error occurs if there's no such binding) is returned. If both*command* and *sequence* are missing then a list of all the e vent sequences for which bindings have been defined for *tagName*.

pathName **element cget** *elemName option*
Returns the current value of the element configuration option given by *option*. *Option* may be any of the options described below for the element **configure** operation.

pathName **element closest** *x y* ?*option value*?... ?*elemName*?...
Finds the data point representing the bar closest to the window coordinates *x* and *y* in the element *elemName*. *ElemName* is the name of an element, which must be displayed. If no elements are specified, then all displayed elements are searched. It returns a list containing the name of the closest element, the index of its closest point, and the graph coordinates of the point. If no data point within the threshold distance can be found, "" is returned. The following *option-value* pairs are available.

−halo *pixels*
Specifies a threshold distance where selected data points are ignored. *Pixels* is a v alid screen distance, such as **2** or **1.2i**. If this option isn't specified, then it defaults to the value of the **barchart**'s **−halo** option.

pathName **element configure** *elemName* ?*elemName*... ?*option value*?...
Queries or modifies the configuration options for elements. Several elements can be modified at the same time. If *option* isn't specified, a list describing all the current options for *elemName* is returned. If*option* is specified, b ut not *value*, then a list describing the option *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the element option *option* is set to *value*. The following options are valid for elements.

−activepen *penName*
Specifies pen to use to draw active element. If *penName* is "", no active elements will be drawn. The default is **activeLine**.

−bindtags *tagList*
Specifies the binding tags for the element. *T agList* is a list of binding tag names. The tags and their order will determine how events for elements. Each tag in the list matching the current event sequence will have its Tcl command executed. Implicitly the name of the element is always the first tag in the list. The default value is **all**.

−background *color*
Sets the the color of the border around each bar. The default is **white**.

−barwidth *value*
Specifies the width the bars drawn for the element. *Value* is the width in X-coordinates. If this option isn't specified, the width of each bar is the value of the widget's **−barwidth** option.

−baseline *value*
Specifies the baseline of the bar segments. This affects how bars are drawn since bars are drawn from their respective y-coordinate the baseline. By default the baseline is **0.0**.

−borderwidth *pixels*
Sets the border width of the 3-D border drawn around the outside of each bar. The **−relief** option determines if such a border is drawn. *Pixels* must be a valid screen distance like **2** or **0.25i**. The default is **2**.

−data *coordList*
Specifies the X−Y coordinates of the data. *Coordlist* is a list of numeric expressions representing the X−Y coordinate pairs of each data point.

−foreground *color*
Sets the color of the interior of the bars.

−hide *boolean*
Indicates whether the element is displayed. The default is **no**.

−label *text*
Sets the element's label in the legend. If *text* is "", the element will have no entry in the legend. The default label is the element's name.

−mapx *xAxis*
Selects the X−axis to map the element's X−coordinates onto. *XAxis* must be the name of an axis. The default is **x**.

−mapy *yAxis*
Selects the Y−axis to map the element's Y−coordinates onto. *YAxis* must be the name of an axis. The default is **y**.

−relief *string*
Specifies the 3-D effect desired for bars. *Relief* indicates how the interior of the bar should appear relative to the surface of the chart; for example, **raised** means the bar should appear to protrude from the surface of the plotting area. The default is **raised**.

−stipple *bitmap*
Specifies a stipple pattern with which to draw the bars. If *bitmap* is "", then the bar is drawn in a solid fashion.

−xdata *xVector*
Specifies the x-coordinate vector of the data. *XVector* is the name of a BLT vector or a list of numeric expressions.

−ydata *yVector*
Specifies the y-coordinate vector of the data. *YVector* is the name of a BLT vector or a list of numeric expressions.

Element configuration options may also be set by the **option** command. The resource names in the option database are prefixed by **elem**.

```
option add *Barchart.Element.background blue
```

pathName **element create** *elemName* ?*option value*?...
Creates a new element *elemName*. Element names must be unique, so an element*elemName* may not already exist. If additional arguments are present, they specify any of the element options valid for element **configure** operation.

pathName **element deactivate** *pattern*...
Deactivates all the elements matching *pattern* for the graph. Elements whose names match any of the patterns given are redrawn using their normal colors.

pathName **element delete** ?*pattern*?...
Deletes all the elements matching *pattern* for the graph. Elements whose names match any of the patterns given are deleted. The graph will be redrawn without the deleted elements.

pathName **element exists** *elemName*
Returns **1** if an element *elemName* currently exists and **0** otherwise.

pathName **element names** ?*pattern*?...
Returns the elements matching one or more pattern. If no *pattern* is given, the names of all elements is returned.

pathName **element show** ?*nameList*?
Queries or modifies the element display list. The element display list designates the elements drawn and in what order. *NameList* is a list of elements to be displayed in the order they are named. If there is no*nameList* ar gument, the current display list is returned.

pathName **element type** *elemName*
Returns the type of *elemName*. If the element is a bar element, the commands returns the string **"bar"**, otherwise it returns **"line"**.

GRID COMPONENT

Grid lines extend from the major and minor ticks of each axis horizontally or vertically across the plotting area. The following operations are available for grid lines.

pathName **grid cget** *option*
Returns the current value of the grid line configuration option given by *option*. *Option* may be any option described below for the grid **configure** operation.

pathName **grid configure** ?*option value*?...
Queries or modifies the configuration options for grid lines. If *option* isn't specified, a list describing all the current grid options for *pathName* is returned. If *option* is specified, but not *value*, then a list describing *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the grid line option *option* is set to *value*. The following options are valid for grid lines.

−color *color*
Sets the color of the grid lines. The default is **black**.

−dashes *dashList*
Sets the dash style of the grid lines. *DashList* is a list of up to 11 numbers that alternately represent the lengths of the dashes and gaps on the grid lines. Each number must be between 1 and 255. If *dashList* is "", the grid will be solid lines.

−hide *boolean*
Indicates whether the grid should be drawn. If *boolean* is true, grid lines are not shown. The default is **yes**.

−linewidth *pixels*
Sets the width of grid lines. The default width is **1**.

−mapx *xAxis*
Specifies the X−axis to display grid lines. *XAxis* must be the name of an axis or "" for no grid lines. The default is "".

−mapy *yAxis*
Specifies the Y−axis to display grid lines. *Y Axis* must be the name of an axis or "" for no grid lines. The default is **y**.

−minor *boolean*
Indicates whether the grid lines should be drawn for minor ticks. If *boolean* is true, the lines will appear at minor tick intervals. The default is **1**.

Grid configuration options may also be set by the **option** command. The resource name and class are **grid** and **Grid** respectively.

```
option add *Barchart.grid.LineWidth 2
option add *Barchart.Grid.Color     black
```

pathName **grid off**
Turns off the display the grid lines.

pathName **grid on**
Turns on the display the grid lines.

pathName **grid toggle**
Toggles the display of the grid.

LEGEND COMPONENT

The legend displays a list of the data elements. Each entry consists of the element's symbol and label. The legend can appear in any margin (the default location is in the right margin). It can also be positioned anywhere within the plotting area.

The following operations are valid for the legend.

pathName **legend activate** *pattern...*
Selects legend entries to be drawn using the active legend colors and relief. All entries whose element names match *pattern* are selected. To be selected, the element name must match only one *pattern*.

pathName **legend bind** *tagName* ?*sequence*? ?*command*?
Associates *command* with *tagName* such that whenever the event sequence given by *sequence* occurs for a legend entry with this tag, *command* will be invoked. Implicitly the element names in the entry are tags. The syntax is similar to the **bind** command except that it operates on legend entries, rather than widgets. See the **bind** manual entry for complete details on *sequence* and the substitutions performed on *command* before invoking it.

If all arguments are specified then a new binding is created, replacing any existing binding for the same *sequence* and *tagName*. If the first character of*command* is **+** then *command* augments an existing binding rather than replacing it. If no *command* argument is provided then the command currently associated with *tagName* and *sequence* (it's an error occurs if there's no such binding) is returned. If both*command* and *sequence* are missing then a list of all the e vent sequences for which bindings have been defined for *tagName*.

pathName **legend cget** *option*
Returns the current value of a legend configuration option. *Option* may be an y option described below in the legend **configure** operation.

pathName **legend configure** ?*option value*?...

Queries or modifies the configuration options for the legend. If*option* isn' t specified, a list describing the current legend options for *pathName* is returned. If *option* is specified, but not *value*, then a list describing *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the legend option *option* is set to *value*. The following options are valid for the legend.

−activebackground *color*

Sets the background color for active legend entries. All legend entries marked active (see the legend **activate** operation) are drawn using this background color.

−activeborderwidth *pixels*

Sets the width of the 3-D border around the outside edge of the active legend entries. The default is **2**.

−activeforeground *color*

Sets the foreground color for active legend entries. All legend entries marked as active (see the legend **activate** operation) are drawn using this foreground color.

−activerelief *relief*

Specifies the 3-D effect desired for active legend entries. *Relief* denotes ho w the interior of the entry should appear relative to the legend; for example, **raised** means the entry should appear to protrude from the legend, relative to the surface of the legend. The default is **flat**.

−anchor *anchor*

Tells how to position the legend relative to the positioning point for the legend. This is dependent on the value of the **−position** option. The default is **center**.

left or **right**	The anchor describes how to position the legend vertically.
top or **bottom**	The anchor describes how to position the legend horizontally.
@x,y	The anchor specifies how to position the legend relative to the positioning point. For example, if *anchor* is **center** then the legend is centered on the point; if *anchor* is **n** then the legend will be drawn such that the top center point of the rectangular region occupied by the legend will be at the positioning point.
plotarea	The anchor specifies how to position the legend relative to the plotting area. For example, if *anchor* is **center** then the legend is centered in the plotting area; if *anchor* is **ne** then the legend will be drawn such that occupies the upper right corner of the plotting area.

−background *color*

Sets the background color of the legend. If *color* is "", the legend background with be transparent.

−bindtags *tagList*

Specifies the binding tags for legend entries. *T agList* is a list of binding tag names. The tags and their order will determine how events for legend entries. Each tag in the list matching the current event sequence will have its Tcl command executed. The default value is **all**.

−borderwidth *pixels*

Sets the width of the 3-D border around the outside edge of the legend (if such border is being drawn; the **relief** option determines this). The default is **2** pixels.

−font *fontName*
FontName specifies a font to use when drawing the labels of each element into the legend. The default is ***-Helvetica-Bold-R-Normal-*-12-120-***.

−foreground *color*
Sets the foreground color of the text drawn for the element's label. The default is **black**.

−hide *boolean*
Indicates whether the legend should be displayed. If *boolean* is true, the legend will not be draw. The default is **no**.

−ipadx *pad*
Sets the amount of internal padding to be added to the width of each legend entry. *Pad* can be a list of one or two screen distances. If *pad* has two elements, the left side of the legend entry is padded by the first distance and the right side by the second. If *pad* is just one distance, both the left and right sides are padded evenly. The default is **2**.

−ipady *pad*
Sets an amount of internal padding to be added to the height of each legend entry. *Pad* can be a list of one or two screen distances. If *pad* has two elements, the top of the entry is padded by the first distance and the bottom by the second. If *pad* is just one distance, both the top and bottom of the entry are padded evenly. The default is **2**.

−padx *pad*
Sets the padding to the left and right exteriors of the legend. *Pad* can be a list of one or two screen distances. If *pad* has two elements, the left side of the legend is padded by the first distance and the right side by the second. If *pad* has just one distance, both the left and right sides are padded evenly. The default is **4**.

−pady *pad*
Sets the padding above and below the legend. *Pad* can be a list of one or two screen distances. If*pad* has tw o elements, the area above the legend is padded by the first distance and the area below by the second. If *pad* is just one distance, both the top and bottom areas are padded evenly. The default is **0**.

−position *pos*
Specifies where the legend is drawn. The **−anchor** option also affects where the legend is positioned. If*pos* is **left**, **left**, **top**, or **bottom**, the le gend is drawn in the specified margin. If*pos* is **plotar ea**, then the legend is drawn inside the plotting area at a particular anchor. If*pos* is in the form "*@x,y*", where *x* and *y* are the windo w coordinates, the legend is drawn in the plotting area at the specified coordinates. The default is **right**.

−raised *boolean*
Indicates whether the legend is above or below the data elements. This matters only if the legend is in the plotting area. If *boolean* is true, the legend will be drawn on top of any elements that may overlap it. The default is **no**.

−relief *relief*
Specifies the 3-D effect for the border around the legend. *Relief* specifies how the interior of the legend should appear relative to the bar chart; for example, **raised** means the legend should appear to protrude from the bar chart, relative to the surface of the bar chart. The default is **sunken**.

Legend configuration options may also be set by the **option** command. The resource name and class are **legend** and **Legend** respectively.

```
option add *Barchart.legend.Foreground blue
option add *Barchart.Legend.Relief     raised
```

pathName **legend deactivate** *pattern...*

Selects legend entries to be drawn using the normal legend colors and relief. All entries whose element names match *pattern* are selected. To be selected, the element name must match only one *pattern*.

pathName **legend get** *pos*

Returns the name of the element whose entry is at the screen position *pos* in the legend. *Pos* must be in the form "*@x,y*", where *x* and *y* are window coordinates. If the given coordinates do not lie over a legend entry, "" is returned.

PEN COMPONENTS

Pens define attributes for elements. Pens mirror the configuration options of data elements that pertain to how symbols and lines are drawn. Data elements use pens to determine how they are drawn. A data element may use several pens at once. In this case, the pen used for a particular data point is determined from each element's weight vector (see the element's **−weight** and **−style** options).

One pen, called **activeBar**, is automatically created. It's used as the default active pen for elements. So you can change the active attributes for all elements by simply reconfiguring this pen.

```
.g pen configure "activeBar" -fg green -bg green4
```

You can create and use several pens. To create a pen, invoke the pen component and its create operation.

```
.g pen create myPen
```

You map pens to a data element using either the element's **−pen** or **−activepen** options.

```
.g element create "e1" -xdata $x -ydata $tempData \
    -pen myPen
```

An element can use several pens at once. This is done by specifying the name of the pen in the element's style list (see the **−styles** option).

```
.g element configure "e1" -styles { myPen 2.0 3.0 }
```

This says that any data point with a weight between 2.0 and 3.0 is to be drawn using the pen **myPen**. All other points are drawn with the element's default attributes.

The following operations are available for pen components.

pathName **pen cget** *penName option*

Returns the current value of the option given by *option* for *penName*. *Option* may be any option described below for the pen **configure** operation.

pathName **pen configure** *penName* ?*penName...* ?*option value*?...

Queries or modifies the configuration options of *penName*. Several pens can be modified at once. If *option* isn't specified, a list describing the current options for *penName* is returned. If *option* is specified, but not *value*, then a list describing *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the pen option *option* is set to *value*. The following options are valid for pens.

−background *color*

Sets the the color of the border around each bar. The default is **white**.

−borderwidth *pixels*

Sets the border width of the 3-D border drawn around the outside of each bar. The **−relief** option determines if such a border is drawn. *Pixels* must be a valid screen distance

like **2** or **0.25i**. The default is **2**.

−foreground *color*
Sets the color of the interior of the bars.

−relief *string*
Specifies the 3-D effect desired for bars. *Relief* indicates ho w the interior of the bar should appear relative to the surface of the chart; for example, **raised** means the bar should appear to protrude from the bar chart, relative to the surface of the plotting area. The default is **raised**.

−stipple *bitmap*
Specifies a stipple pattern with which to draw the bars. If *bitmap* is "", then the bar is drawn in a solid fashion.

−type *elemType*
Specifies the type of element the pen is to be used with. This option should only be employed when creating the pen. This is for those that wish to mix different types of elements (bars and lines) on the same graph. The default type is "bar".

Pen configuration options may be also be set by the **option** command. The resource class is **Pen**. The resource names are the names of the pens.

```
option add *Barchart.Pen.Foreground   blue
option add *Barchart.activeBar.foreground  green
```

pathName **pen create** *penName* ?*option value*?...
Creates a new pen by the name *penName*. No pen by the same name can already exist. *Option* and *value* are described in above in the pen **configure** operation.

pathName **pen delete** ?*penName*?...
Deletes the named pens. A pen is not really deleted until it is not longer in use, so it's safe to delete pens mapped to elements.

pathName **pen names** ?*pattern*?...
Returns a list of pens matching zero or more patterns. If no *pattern* argument is give, the names of all pens are returned.

POSTSCRIPT COMPONENT

The barchart can generate encapsulated PostScript output. There are several configuration options you can specify to control how the plot will be generated. You can change the page dimensions and borders. The plot itself can be scaled, centered, or rotated to landscape. The PostScript output can be written directly to a file or returned through the interpreter.

The following postscript operations are available.

pathName **postscript cget** *option*
Returns the current value of the postscript option given by *option*. *Option* may be any option described below for the postscript **configure** operation.

pathName **postscript configure** ?*option value*?...
Queries or modifies the configuration options for PostScript generation. If *option* isn't specified, a list describing the current postscript options for *pathName* is returned. If *option* is specified, but not *value*, then a list describing *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the postscript option *option* is set to *value*. The following postscript options are available.

−center *boolean*

Indicates whether the plot should be centered on the PostScript page. If *boolean* is false, the plot will be placed in the upper left corner of the page. The default is **1**.

−colormap *varName*

VarName must be the name of a global array variable that specifies a color mapping from the X color name to PostScript. Each element of *varName* must consist of PostScript code to set a particular color value (e.g. "**1.0 1.0 0.0 setrgbcolor**"). When generating color information in PostScript, the array variable *varName* is checked if an element of the name as the color exists. If so, it uses its value as the PostScript command to set the color. If this option hasn't been specified, or if there isn't an entry in *varName* for a given color, then it uses the red, green, and blue intensities from the X color.

−colormode *mode*

Specifies how to output color information. *Mode* must be either **color** (for full color output), **gray** (convert all colors to their gray-scale equivalents) or **mono** (convert foreground colors to black and background colors to white). The default mode is **color**.

−fontmap *varName*

VarName must be the name of a global array variable that specifies a font mapping from the X font name to PostScript. Each element of *varName* must consist of a Tcl list with one or two elements; the name and point size of a PostScript font. When outputting PostScript commands for a particular font, the array variable *varName* is checked to see if an element by the specified font exists. If there is such an element, then the font information contained in that element is used in the PostScript output. (If the point size is omitted from the list, the point size of the X font is used). Otherwise the X font is examined in an attempt to guess what PostScript font to use. This works only for fonts whose foundry property is *Adobe* (such as Times, Helvetica, Courier, etc.). If all of this fails then the font defaults to **Helvetica-Bold**.

−decorations *boolean*

Indicates whether PostScript commands to generate color backgrounds and 3-D borders will be output. If *boolean* is false, the graph will background will be white and no 3-D borders will be generated. The default is **1**.

−height *pixels*

Sets the height of the plot. This lets you print the bar chart with a height different from the one drawn on the screen. If *pixels* is 0, the height is the same as the widget's height. The default is **0**.

−landscape *boolean*

If *boolean* is true, this specifies the printed area is to be rotated 90 degrees. In non-rotated output the X−axis of the printed area runs along the short dimension of the page (''portrait'' orientation); in rotated output the X−axis runs along the long dimension of the page (''landscape'' orientation). Defaults to **0**.

−maxpect *boolean*

Indicates to scale the plot so that it fills the PostScript page. The aspect ratio of the barchart is still retained. The default is **0**.

−padx *pad*

Sets the horizontal padding for the left and right page borders. The borders are exterior to the plot. *P ad* can be a list of one or two screen distances. If *pad* has two elements, the left border is padded by the first distance and the right border by the second. If *pad* has just one distance, both the left and right borders are padded evenly. The default is **1i**.

−pady *pad*
Sets the vertical padding for the top and bottom page borders. The borders are exterior to the plot. *P ad* can be a list of one or two screen distances. If *pad* has two elements, the top border is padded by the first distance and the bottom border by the second. If *pad* has just one distance, both the top and bottom borders are padded evenly. The default is **1i**.

−paperheight *pixels*
Sets the height of the postscript page. This can be used to select between different page sizes (letter, A4, etc). The default height is **11.0i**.

−paperwidth *pixels*
Sets the width of the postscript page. This can be used to select between different page sizes (letter, A4, etc). The default width is **8.5i**.

−width *pixels*
Sets the width of the plot. This lets you generate a plot of a width different from that of the widget. If *pixels* is 0, the width is the same as the widget's width. The default is **0**.

Postscript configuration options may be also be set by the **option** command. The resource name and class are **postscript** and **Postscript** respectively.

```
option add *Barchart.postscript.Decorations false
option add *Barchart.Postscript.Landscape   true
```

pathName **postscript output** ?*fileName*? ?*option value*?...
Outputs a file of encapsulated PostScript. If a *fileName* argument isn't present, the command returns the PostScript. If any *option-value* pairs are present, they set configuration options controlling how the PostScript is generated. *Option* and *value* can be an ything accepted by the postscript **configure** operation above.

MARKER COMPONENTS

Markers are simple drawing procedures used to annotate or highlight areas of the graph. Markers have various types: text strings, bitmaps, images, connected lines, windows, or polygons. They can be associated with a particular element, so that when the element is hidden or un-hidden, so is the marker. By default, markers are the last items drawn, so that data elements will appear in behind them. You can change this by configuring the **−under** option.

Markers, in contrast to elements, don't affect the scaling of the coordinate axes. They can also have *elastic* coordinates (specified by **-Inf** and **Inf** respectively) that translate into the minimum or maximum limit of the axis. For example, you can place a marker so it always remains in the lower left corner of the plotting area, by using the coordinates **-Inf**,**-Inf**.

The following operations are available for markers.

pathName **marker after** *markerId* ?*afterId*?
Changes the order of the markers, drawing the first marker after the second. If no second *afterId* argument is specified, the marker is placed at the end of the display list. This command can be used to control how markers are displayed since markers are drawn in the order of this display list.

pathName **marker before** *markerId* ?*beforeId*?
Changes the order of the markers, drawing the first marker before the second. If no second *beforeId* argument is specified, the marker is placed at the beginning of the display list. This command can be used to control how markers are displayed since markers are drawn in the order of this display list.

pathName **marker bind** *tagName* ?*sequence*? ?*command*?
Associates *command* with *tagName* such that whenever the event sequence given by *sequence* occurs for a marker with this tag, *command* will be invoked. The syntax is similar to the**bind**

command except that it operates on graph markers, rather than widgets. See the **bind** manual entry for complete details on *sequence* and the substitutions performed on *command* before invoking it.

If all arguments are specified then a new binding is created, replacing any existing binding for the same *sequence* and *tagName*. If the first character of*command* is + then *command* augments an existing binding rather than replacing it. If no *command* argument is provided then the command currently associated with *tagName* and *sequence* (it's an error occurs if there's no such binding) is returned. If both*command* and *sequence* are missing then a list of all the e vent sequences for which bindings have been defined for *tagName*.

pathName **marker cget** *option*

Returns the current value of the marker configuration option given by *option*. *Option* may be any option described below in the **configure** operation.

pathName **marker configure** *markerId* ?*option value*?...

Queries or modifies the configuration options for markers. If*option* isn' t specified, a list describing the current options for *markerId* is returned. If *option* is specified, but not *value*, then a list describing *option* is returned. If one or more *option* and *value* pairs are specified, then for each pair, the marker option *option* is set to *value*.

The following options are valid for all markers. Each type of marker also has its own type-specific options. They are described in the sections below.

−bindtags *tagList*

Specifies the binding tags for the marker. *TagList* is a list of binding tag names. The tags and their order will determine how events for markers are handled. Each tag in the list matching the current event sequence will have its Tcl command executed. Implicitly the name of the marker is always the first tag in the list. The default value is **all**.

−coords *coordList*

Specifies the coordinates of the marker. *CoordList* is a list of graph coordinates. The number of coordinates required is dependent on the type of marker. Text, image, and window markers need only two coordinates (an X−Y coordinate). Bitmap markers can take either two or four coordinates (if four, they represent the corners of the bitmap). Line markers need at least four coordinates, polygons at least six. If *coordList* is "", the marker will not be displayed. The default is "".

−element *elemName*

Links the marker with the element *elemName*. The marker is drawn only if the element is also currently displayed (see the element's **show** operation). If *elemName* is "", the marker is always drawn. The default is "".

−hide *boolean*

Indicates whether the marker is drawn. If *boolean* is true, the marker is not drawn. The default is **no**.

−mapx *xAxis*

Specifies the X−axis to map the marker's X−coordinates onto. *XAxis* must the name of an axis. The default is **x**.

−mapy *yAxis*

Specifies the Y−axis to map the marker's Y−coordinates onto. *YAxis* must the name of an axis. The default is **y**.

−name *markerId*

Changes the identifier for the marker. The identifier*mark erId* can not already be used by another marker. If this option isn't specified, the marker's name is uniquely generated.

−under *boolean*
Indicates whether the marker is drawn below/above data elements. If *boolean* is true, the marker is be drawn underneath the data elements. Otherwise, the marker is drawn on top of the element. The default is **0**.

−xoffset *pixels*
Specifies a screen distance to offset the marker horizontally. *Pixels* is a valid screen distance, such as **2** or **1.2i**. The default is **0**.

−yoffset *pixels*
Specifies a screen distance to offset the markers vertically. *Pixels* is a valid screen distance, such as **2** or **1.2i**. The default is **0**.

Marker configuration options may also be set by the **option** command. The resource class is either **BitmapMarker**, **ImageMarker**, **LineMarker**, **PolygonMarker**, **TextMarker**, or **WindowMarker**, depending on the type of marker. The resource name is the name of the marker.

```
option add *Barchart.TextMarker.Foreground white
option add *Barchart.BitmapMarker.Foreground white
option add *Barchart.m1.Background     blue
```

pathName **marker create** *type* ?*option value*?...
Creates a marker of the selected type. *Type* may be either **text**, **line**, **bitmap**, **image**, **polygon**, or **window**. This command returns the marker identifier, used as the *markerId* argument in the other marker-related commands. If the **−name** option is used, this overrides the normal marker identifier. If the name provided is already used for another marker, the new marker will replace the old.

pathName **marker delete** ?*name*?...
Removes one of more markers. The graph will automatically be redrawn without the marker..

pathName **marker exists** *markerId*
Returns **1** if the marker *markerId* exists and **0** otherwise.

pathName **marker names** ?*pattern*?
Returns the names of all the markers that currently exist. If*pattern* is supplied, only those mark ers whose names match it will be returned.

pathName **marker type** *markerId*
Returns the type of the marker given by *markerId*, such as **line** or **text**. If*mark erId* is not a valid a marker identifier, `""` is returned.

BITMAP MARKERS

A bitmap marker displays a bitmap. The size of the bitmap is controlled by the number of coordinates specified. If two coordinates, they specify the position of the top-left corner of the bitmap. The bitmap retains its normal width and height. If four coordinates, the first and second pairs of coordinates represent the corners of the bitmap. The bitmap will be stretched or reduced as necessary to fit into the bounding rectangle.

Bitmap markers are created with the marker's **create** operation in the form: *pathName* **marker create bitmap** ?*option value*?... There may be many *option-value* pairs, each sets a configuration options for the marker. These same*option−value* pairs may be used with the mark er's **configure** operation.

The following options are specific to bitmap markers:

−background *color*
Same as the **−fill** option.

−bitmap *bitmap*
Specifies the bitmap to be displayed. If *bitmap* is "", the marker will not be displayed. The default is "".

−fill *color*
Sets the background color of the bitmap. If *color* is the empty string, no background will be transparent. The default background color is "".

−foreground *color*
Same as the **−outline** option.

−mask *mask*
Specifies a mask for the bitmap to be displayed. This mask is a bitmap itself, denoting the pixels that are transparent. If *mask* is "", all pixels of the bitmap will be drawn. The default is "".

−outline *color*
Sets the foreground color of the bitmap. The default value is **black**.

−rotate *theta*
Sets the rotation of the bitmap. *Theta* is a real number representing the angle of rotation in degrees. The marker is first rotated and then placed according to its anchor position. The default rotation is **0.0**.

IMAGE MARKERS

A image marker displays an image. Image markers are created with the marker's **create** operation in the form: *pathName* **marker create image** ?*option value*?... There may be many *option-value* pairs, each sets a configuration option for the marker. These same*option−value* pairs may be used with the mark er's **configure** operation.

The following options are specific to image markers:

−anchor *anchor*
Anchor tells how to position the image relative to the positioning point for the image. For example, if *anchor* is **center** then the image is centered on the point; if *anchor* is **n** then the image will be drawn such that the top center point of the rectangular region occupied by the image will be at the positioning point. This option defaults to **center**.

−image *image*
Specifies the image to be drawn. If*ima ge* is "", the marker will not be drawn. The default is "".

LINE MARKERS

A line marker displays one or more connected line segments. Line markers are created with marker's **create** operation in the form: *pathName* **marker create line** ?*option value*?... There may be many *option-value* pairs, each sets a configuration option for the marker. These same*option-value* pairs may be used with the marker's **configure** operation.

The following options are specific to line markers:

−dashes *dashList*
Sets the dash style of the line. *DashList* is a list of up to 11 numbers that alternately represent the lengths of the dashes and gaps on the line. Each number must be between 1 and 255. If *dashList* is "", the marker line will be solid.

−fill *color*
Sets the background color of the line. This color is used with striped lines (see the **−fdashes** option). If*color* is the empty string, no background color is dra wn (the line will be dashed, not striped). The default background color is "".

−linewidth *pixels*
Sets the width of the lines. The default width is **0**.

−outline *color*
Sets the foreground color of the line. The default value is **black**.

−stipple *bitmap*
Specifies a stipple pattern used to draw the line, rather than a solid line. *Bitmap* specifies a bitmap to use as the stipple pattern. If *bitmap* is "", then the line is drawn in a solid fashion. The default is "".

POLYGON MARKERS

A polygon marker displays a closed region described as two or more connected line segments. It is assumed the first and last points are connected. Polygon markers are created using the marker **create** operation in the form: *pathName* **marker create polygon** ?*option value*?... There may be many *option-value* pairs, each sets a configuration option for the marker. These same*option−value* pairs may be used with the **marker configure** command to change the marker's configuration. The following options are supported for polygon markers:

−dashes *dashList*
Sets the dash style of the outline of the polygon. *DashList* is a list of up to 11 numbers that alternately represent the lengths of the dashes and gaps on the outline. Each number must be between 1 and 255. If *dashList* is "", the outline will be a solid line.

−fill *color*
Sets the fill color of the polygon. If *color* is "", then the interior of the polygon is transparent. The default is **white**.

−linewidth *pixels*
Sets the width of the outline of the polygon. If *pixels* is zero, no outline is drawn. The default is **0**.

−outline *color*
Sets the color of the outline of the polygon. If the polygon is stippled (see the **−stipple** option), then this represents the foreground color of the stipple. The default is **black**.

−stipple *bitmap*
Specifies that the polygon should be drawn with a stippled pattern rather than a solid color. *Bitmap* specifies a bitmap to use as the stipple pattern. If *bitmap* is "", then the polygon is filled with a solid color (if the **−fill** option is set). The default is "".

TEXT MARKERS

A text marker displays a string of characters on one or more lines of text. Embedded newlines cause line breaks. They may be used to annotate regions of the graph. Text markers are created with the **create** operation in the form: *pathName* **marker create text** ?*option value*?... There may be many *option-value* pairs, each sets a configuration option for the text marker. These same*option−value* pairs may be used with the marker's **configure** operation.

The following options are specific to text markers:

−anchor *anchor*
Anchor tells how to position the text relative to the positioning point for the text. For example, if *anchor* is **center** then the text is centered on the point; if *anchor* is **n** then the text will be drawn such that the top center point of the rectangular region occupied by the text will be at the positioning point. This default is **center**.

−background *color*
Same as the **−fill** option.

−font *fontName*
Specifies the font of the text. The default is ***-Helvetica-Bold-R-Normal-*-120-***.

−fill *color*
Sets the background color of the text. If*color* is the empty string, no background will be transpar - ent. The default background color is "".

−foreground *color*
Same as the **−outline** option.

−justify *justify*
Specifies how the text should be justified. This matters only when the marker contains more than one line of text. *Justify* must be **left**, **right**, or **center**. The default is **center**.

−outline *color*
Sets the color of the text. The default value is **black**.

−padx *pad*
Sets the padding to the left and right exteriors of the text. *Pad* can be a list of one or two screen distances. If*pad* has tw o elements, the left side of the text is padded by the first distance and the right side by the second. If *pad* has just one distance, both the left and right sides are padded evenly. The default is **4**.

−pady *pad*
Sets the padding above and below the text. *Pad* can be a list of one or two screen distances. If *pad* has two elements, the area above the text is padded by the first distance and the area below by the second. If*pad* is just one distance, both the top and bottom areas are padded e venly. The default is **4**.

−rotate *theta*
Specifies the number of degrees to rotate the text. *Theta* is a real number representing the angle of rotation. The marker is first rotated along its center and is then drawn according to its anchor position. The default is **0.0**.

−text *text*
Specifies the text of the marker. The exact way the text is displayed may be affected by other options such as **−anchor** or **−rotate**.

WINDOW MARKERS

A window marker displays a widget at a given position. Window markers are created with the marker's **create** operation in the form: *pathName* **marker create window** ?*option value*?... There may be many *option-value* pairs, each sets a configuration option for the marker. These same*option−value* pairs may be used with the marker's **configure** command.

The following options are specific to window markers:

−anchor *anchor*
Anchor tells how to position the widget relative to the positioning point for the widget. For example, if *anchor* is **center** then the widget is centered on the point; if *anchor* is **n** then the widget will be displayed such that the top center point of the rectangular region occupied by the widget will be at the positioning point. This option defaults to **center**.

−height *pixels*
Specifies the height to assign to the marker's window. If this option isn't specified, or if it is specified as "", then the window is given whatever height the widget requests internally.

−width *pixels*
Specifies the width to assign to the marker's window. If this option isn't specified, or if it is specified as "", then the window is given whatever width the widget requests internally.

−window *pathName*
Specifies the widget to be managed by the barchart. *P athName* must be a child of the **barchart** widget.

GRAPH COMPONENT BINDINGS

Specific barchart components, such as elements, markers and legend entries, can have a command trigger when event occurs in them, much like canvas items in Tk's canvas widget. Not all event sequences are valid. The only binding events that may be specified are those related to the mouse and keyboard (such as **Enter**, **Leave**, **ButtonPress**, **Motion**, and **KeyPress**).

Only one element or marker can be picked during an event. This means, that if the mouse is directly over both an element and a marker, only the uppermost component is selected. This isn't true for legend entries. Both a legend entry and an element (or marker) binding commands will be invoked if both items are picked.

It is possible for multiple bindings to match a particular event. This could occur, for example, if one binding is associated with the element name and another is associated with one of the element's tags (see the **−bindtags** option). When this occurs, all of the matching bindings are invoked. A binding associated with the element name is invoked first, followed by one binding for each of the element's bindtags. If there are multiple matching bindings for a single tag, then only the most specific binding is invoked. A continue command in a binding script terminates that script, and a break command terminates that script and skips any remaining scripts for the event, just as for the bind command.

The **−bindtags** option for these components controls addition tag names which can be matched. Implicitly elements and markers always have tags matching their names. Setting the value of the **−bindtags** option doesn't change this.

C LANGUAGE API

You can manipulate data elements from the C language. There may be situations where it is too expensive to translate the data values from ASCII strings. Or you might want to read data in a special file format.

Data can manipulated from the C language using BLT vectors. You specify the X-Y data coordinates of an element as vectors and manipulate the vector from C. The barchart will be redrawn automatically after the vectors are updated.

From Tcl, create the vectors and configure the element to use them.

```
vector X Y
.g element configure line1 -xdata X -ydata Y
```

To set data points from C, you pass the values as arrays of doubles using the **Blt_ResetVector** call. The vector is reset with the new data and at the next idle point (when Tk re-enters its event loop), the graph will be redrawn automatically.

```
#include <tcl.h>
#include <blt.h>

register int i;
Blt_Vector *xVec, *yVec;
double x[50], y[50];

/* Get the BLT vectors "X" and "Y" (created above from Tcl) */
if ((Blt_GetVector(interp, "X", 50, &xVec) != TCL_OK) ||
    (Blt_GetVector(interp, "Y", 50, &yVec) != TCL_OK)) {
    return TCL_ERROR;
}

for (i = 0; i < 50; i++) {
    x[i] = i * 0.02;
    y[i] = sin(x[i]);
}
```

```
/* Put the data into BLT vectors */
if ((Blt_ResetVector(xVec, x, 50, 50, TCL_VOLATILE) != TCL_OK) ||
    (Blt_ResetVector(yVec, y, 50, 50, TCL_VOLATILE) != TCL_OK)) {
   return TCL_ERROR;
}
```

See the **vector** manual page for more details.

SPEED TIPS

There may be cases where the bar chart needs to be drawn and updated as quickly as possible. If drawing speed becomes a big problem, here are a few tips to speed up displays.

- Try to minimize the number of data points. The more data points looked at, the more work the bar chart must do.
- If your data is generated as floating point values, the time required to convert the data values to and from ASCII strings can be significant, especially when there any many data points. You can avoid the redundant string-to-decimal conversions using the C API to BLT vectors.
- Don't stipple or dash the element. Solid bars are much faster.
- If you update data elements frequently, try turning off the widget's **−bufferelements** option. When the bar chart is first displayed, it draws data elements into an internal pixmap. The pixmap acts as a cache, so that when the bar chart needs to be redrawn again, and the data elements or coordinate axes haven't changed, the pixmap is simply copied to the screen. This is especially useful when you are using markers to highlight points and regions on the bar chart. But if the bar chart is updated frequently, changing either the element data or coordinate axes, the buffering becomes redundant.

LIMITATIONS

Auto-scale routines do not use requested min/max limits as boundaries when the axis is logarithmically scaled.

The PostScript output generated for polygons with more than 1500 points may exceed the limits of some printers (See PostScript Language Reference Manual, page 568). The work-around is to break the polygon into separate pieces.

KEYWORDS

bar chart, widget