GINOGRAF

user guide

version 6.0

BRADLY ASSOCIATES LTD
Manhattan House
140 High Street
Crowthorne
Berkshire RG45 7AY
England
Tel: +44 (1344) 779381
Fax: +44 (1344) 773168
support@gino-graphics.com
www.gino-graphics.com

Contents

Chapter 1

INTRODUCTION

This section covers the scope of the GINOGRAF library and its interface with GINO.

The Scope of GINOGRAF

GINOGRAF is a routine library for displaying statistical data in a graphical form. It is used in conjunction with GINO, which provides the device drivers and graphical primitives required by GINOGRAF. Both packages are almost totally machine independent.

GINOGRAF provides the following facilities :

Graphs

X-Y graphs (lines, splines, curves, symbols), area graphs, error bars, hi-lo graphs, scatter diagrams, square wave graphs.

Charts

Histograms, bar charts, gantt charts, time planners, area charts, step charts, stacked histograms.

Vector Charts

Air/fluid flow diagrams, stress flow, 5D data display.

Polar Charts

Circular axes, polar coordinates.

Pie Charts

Complete Pie Charts, individual segments.

Text Charts

Automated columns of strings, values, lines, symbols and fill styles.

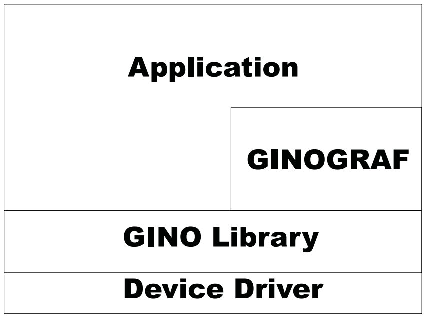
Utility routines

Reference lines, data fitting, arrows, coordinate conversion.

Within most chapters, two levels of routines are provided. Complete drawing routines to present your data in the required form in one easy call, and low level component routines to allow the user to construct comprehensive graphs in a modular way, giving much finer control over positioning and annotation. Complete drawing routines are provided for Graphs, Bar Charts, Histograms, Step Charts, Area Charts, Polar Charts and Pie Charts.

Interfacing With GINO

The GINOGRAF library uses GINO routines to carry out all its drawing and so the availability of the GINO library is paramount to users of GINOGRAF. In addition, any GINOGRAF application will require some GINO routines for certain basic routines, such as device initialization and termination as this is not carried out within GINOGRAF. The relationship between the components of a GINOGRAF application is as follows:



As the GINO library is required for any GINOGRAF application, it follows that the complete functionality of GINO is also available to the GINOGRAF user and the two sets of routines may be used without restriction. For the first-time user of GINOGRAF only graphic device control will be required as described above, but as the application develops, line, colour, character and window control may be added. This chapter covers both the essential routines required by all GINOGRAF applications as well as other areas which can be used to affect the output from GINOGRAF.

Colour, Line Attributes and Filling

Most line drawing done within GINOGRAF is done using the current line and colour attributes as set by GINO. The current colour and line attributes are controlled by routines such as:

`gSetLineColour(col)`	Line colour
`gSetLineWidth(width)`	Line width
`gSetLineEnd(end)`	Line end
`gSetBrokenLine(brk)`	Broken line type

The exceptions are where line and fill indexes are supplied to a GINOGRAF routine by argument, such as the Graph and Chart filling, Pie Chart filling and some Text Chart routines. The default styles are shown in Appendix A of this manual but these may be changed using the following GINO routines:

gDefineLineStyle(line,vis,brk,col,width,type,end)

gDefineBrokenLineStyle(brk,mode,repeat,dash,dot)

gDefineHatchStyle(fill,pitch,angle,xshift,yshift,xshear,xhatch)

Details of these and other routines controlling colour, line style and filling are found in the Line Attributes and Area Filling sections in the GINO User Guide.

Characters and Fonts

By default GINOGRAF uses software-transformable characters to ensure that if transformations (shift, rotate, scale and shear) have been applied within a program using GINOGRAF, the annotation and titling are modified in the same way as the graph itself. This is carried out by an internal call to gSetCharTransformMode(GON) (a GINO routine) which prevents the output of hardware characters by GINOGRAF. In addition, it may result in a different appearance of GINO character strings due to the differences between software characters and hardware characters produced on different devices.

If software characters are required throughout a GINO/GINOGRAF application this can be achieved by calling gSetSoftChars() or gSetCharTransformMode() at the start of the program. Alternatively, if the user requires hardware characters to be output by GINOGRAF, it is possible to prevent the switch to software-transformable characters by using the routine ggSetGraphCharMode():

ggSetGraphCharMode(sw)

where, if **sw** = GGINOMODE the current character mode will be used within GINOGRAF. Users are warned that unpredictable result may occur if ggSetGraphCharMode() is called while transformations are current.

GINOGRAF uses the current character, font and string attributes set by GINO except for string angle (gSetStrAngle()) and justification (gSetStrJustify()) which are always set as required by the relevant GINOGRAF routine. The following GINO routines will however, affect the size and form of graph annotation and labelling:

`gSetCharSize(width,height)`	Character size
`gSetItalicAngle(ang)`	Italic angle
`gSetStrUnderscore(und)`	Underlining
`gSetCharFont(font)`	Font
`gSetFontWeight(weight)`	Font weight
`gSetFontFillStyle(style)`	Font style and filling

In all cases however, the current GINO character mode and attributes are restored at the end of a GINOGRAF routine which has changed them.

GINOGRAF fully caters for the use of all the GINO character string escape facilities which provide for font changes, exponents and indices, italic angles, emboldening and strings that contain more than one line of text. All these features may therefore be used for graph and Pie Chart annotation and titling. A summary of the features is listed below:

`'*F`*nnn*`'`	Change to font nnn	(ie: change to font 3: *F003)
`'*N'`	Move to next line of text block	
`'*E'`	Set exponent	(0.6*height above baseline)
`'*I'`	Set Index	(0.3*height below baseline)
`'*S'`	Underline following characters	
`'*\'`	Set italic -15 deg	
`'*|'`	Set italic 0 deg	
`'*/'`	Set italic +15 degrees	
`'*W'`	Bold following characters	
`'**'`	Output the escape character	
`'*.'`	String terminator	

Full details of all the character routines and features are given in the CHARACTERS section of the GINO User Guide.

Transformations, Windowing and Masking

All output by GINOGRAF is subject to the current GINO transformation, windowing and masking state. As indicated above, to ensure graph annotation is correctly aligned with the graph itself, software-transformable characters are output by default. The control of transformations in GINO allows graphs to be shifted, rotated, scaled or sheared in 2D or 3D as required.

In the case of the complete drawing routines, the current windowing limits affects the position of Graph axes, Pie and Polar Charts. If transformations are also current, the graph can be transformed out of the clipping area resulting in only part of or none of the graph being visible. Users must therefore be careful when using any transformations with the complete drawing routines.

However, masking, whether it be rectangular or polygonal can be used to good effect with GINOGRAF, masking out areas for the placement of text boxes or other annotation. The output is suitable for all graphics devices as no over-drawing is done.

Initializing GINOGRAF

GINOGRAF is automatically initialized when the first GINOGRAF routine is called. This outputs the GINOGRAF banner, initializes all internal variables and enquires the current GINO window limits and character attributes.

The GINOGRAF banner gives the version number of the library being used as well as the copyright message. The banner cannot be suppressed, but it can be forced to appear at a convenient place (for example, the very beginning of a program run), by forcing initialization in a controlled way. This can be achieved by calling the routine

```
ggRestoreAxesSettings().
```

ggRestoreAxesSettings() must however, be called after a GINO device is nominated. ggRestoreAxesSettings() sets up the default position, size and scaling for both axes; as this is done by the first GINOGRAF routine to be called in a program, whatever it is, the call to ggRestoreAxesSettings() does not change the operation of subsequent GINOGRAF routines.

Where graphical output is being produced on the same terminal that the program is being run from, the code:

```
ggRestoreAxesSettings();
gNewDrawing();
```

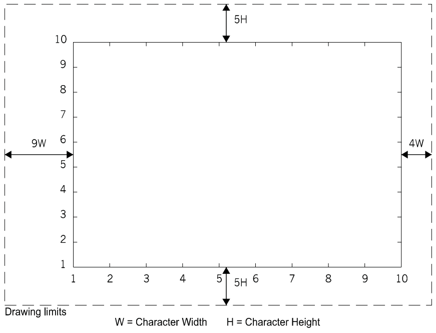
```
call ggRestoreAxesSettings
call gNewDrawing
```

displays the GINOGRAF banner and leaves the screen clear for drawing graphs.

Drawing Area

By default the GINOGRAF drawing area is the same as the current GINO window limits.

This is either the current device or paper limits (as set by the GINO routine gSetDrawingLimits()) or that set by the GINO routine gSetWindow2D() (although the 2D limits from the 3D window routine has the same effect on GINOGRAF). As the window routines also control clipping limits, it may be desirable to set a different GINOGRAF drawing area to be either larger or smaller than the clipping limits. This is achieved through the routine ggSetPlotFrame():



Default position and limits of the graphical axes

ggSetPlotFrame(limits)

where the structure **limits** defines the required limits.

The current drawing limits can be enquired using the routine ggEnqPlotFrame() which also indicates if they were set by ggSetPlotFrame() or the GINO window routines through the returned argument **flg**.

ggEnqPlotFrame(flg,limits)

All the complete Graph and Chart routines use the current drawing area limits to define the default position of the graph or chart. The default positions of graph axes are shown above, where the space around the axes is calculated using the current GINO character size. The default Polar or Pie Chart position is shown above.

Multiple graphs or charts may be positioned on the device or paper area by calling ggSetPlotFrame() with appropriate limits. This is explained in more detail later.

These positions are only used for the complete Graph or Chart routines within GINOGRAF and do not preclude the manual positioning of axes with the routine ggSetAxesPos(), or Polar Charts with the routine ggSetPolarChartAttribs() or Pie Charts with the routine ggSetPieChartFrame().

The drawing area limits can be reset to match the GINO window limits by calling ggRestoreAxesSettings() or ggRestorePieChartSettings().

Graphical Axis Coordinate System

Once GINOGRAF is initialized, there are two coordinate systems available to the GINOGRAF user, user space coordinates and graphical axis coordinates.

User space coordinates represent the picture coordinates, with the origin (0,0) at the bottom left corner of the picture, whether on paper or screen. This coordinate system is used by any GINO routine called directly by the user.

Graphical axis coordinates define points within a Graph, Chart or Polar Chart according to the axis position and scaling that was defined using the axis definition routines ggSetAxesPos() and ggSetAxesScaling(), or any of the complete drawing routines ggPlotGraph(), ggPlotXYPolarChart(). This greatly simplifies the plotting of graphs or charts as the user does not have to compensate for the position of the graph on the paper, redefine the user space origin, or scale values when using logarithmic axes.

Conversion between the two systems is provided by two utility routines ggTransformGraphPoint() and ggTransformSpacePoint().

Both user space coordinates and graphical axis coordinates are affected by the shift, rotate, scale and shear transformations that are provided by GINO. The effect of these transformations may be switched off or on at any time (eg, for outputting text). Full details of transformation and transformation control are given in the WINDOWING & MASKING and TRANSFORMATION CONTROL sections of the GINO User Guide.

Chapter 2

AXES

Introduction to Axes

This chapter concerns itself with the definition, display and labelling of standard two dimensional graph axes. The definition of an axes system for the following graph and chart chapters is required to set up an appropriate graphical coordinate system in which to display the data. As well as the default numerical labelling of axes, textual labelling is covered together with fine control of the position and format of graphical annotation.

Summary of the functionality of the axes routines:

`ggSetAxesPos()` `ggEnqAxesPos()`	Set/enquire axis position
`ggSetAxesScaling()` `ggEnqAxesScaling()`	Set/enquire axes range and scaling type
`ggDrawAxes()`	Draws axes, tick marks and numeric annotation
`ggAddGrid()`	Draws a stylized frame
`ggSetGridMarker()` `ggEnqGridMarker()`	Set/enquire grid intersection symbol
`ggSetAxesAnnotation()` `ggEnqAxesAnnotation()`	Set/enquire numeric format of axes annotation
`ggDrawAxesTitle()`	Output an axis title
`ggDrawAxesLabels()`	Axes labelling
`ggSetAxesAttribs()` `ggEnqAxesAttribs()`	Set/enquire axes annotation attributes
`ggSetDateFormat()` `ggEnqDateFormat()`	Set/enquire date input format
`ggSetDateAxesScaling()` `ggEnqDateAxesScaling()`	Set/enquire date axes scaling
`ggSetDateAxesAnnotation()` `ggEnqDateAxesAnnotation()`	Set/enquire date axes output format
`ggConvertDates()`	Converts date data

`ggConvertDateToGraph()`	Converts date into a real value representing a day number
`ggConvertGraphToDate()`	Converts a date value into a character string in the current input date format
`ggRestoreAxesSettings()`	Restore default axes settings

Axes Definition

The two routines for setting up the axes system are:

`ggSetAxesPos()`	for positioning each axis
`ggSetAxesScaling()`	for setting the ranges and scale type

There are default positions and data ranges for both axes, but these are only provided for a fail safe situation. While the default position may be satisfactory, it is essential to define the scale type and data ranges to both X and Y axes (using ggSetAxesScaling()) before drawing any graph form. The drawing and titling of axes is optional.

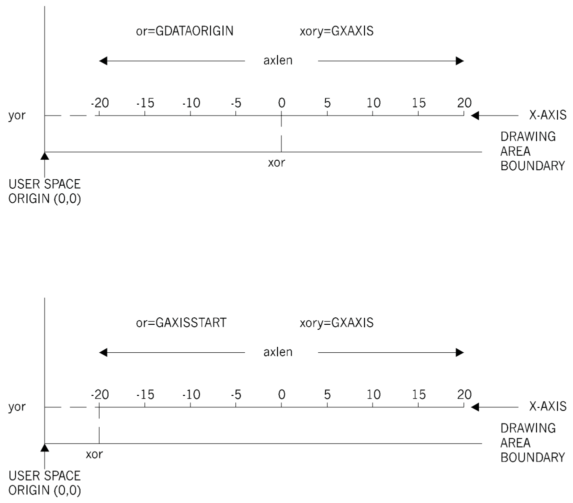
Axis Positioning

The routine call which positions each axis is ggSetAxesPos():

```
ggSetAxesPos(or,xor,yor,axlen,xory)
```

where **xory** determines which axis position is being defined (**xory** = GXAXIS for X axis and **xory** = GYAXIS for Y axis). **xor**, **yor** determines the axis position in user space coordinates and **axlen** determines the length in current units. The position (**xor**, **yor**) is defined as either the point where the axis starts (**or**=GAXISSTART) or the point where the natural origin of the data (the value 0.0) occurs on the axis (**or**=GDATAORIGIN).

The relationship of the above arguments is illustrated below



Relationship of ggSetAxesPos arguments

For the two axes to intersect at **xor**, **yor** then the values for **or**, **xor** and **yor** should be constant for the calls of ggSetAxesPos() for each axis.

Successive calls to ggSetAxesPos() override earlier definitions for the same axis.

Axis Scaling

The routine to set the scaling type and range of either axis is ggSetAxesScaling():

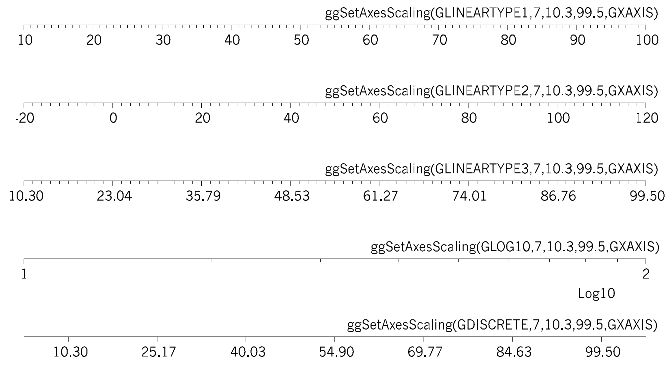
```
ggSetAxesScaling(scale,nints,vbeg,vend,xory)
```

where **xory** determines which axis scaling is being defined (**xory** = GXAXIS for X axis and **xory** = GYAXIS for Y axis). **nints** is the required number of intervals and **vbeg**,**vend** specify the start and end of the axis range.

There are five different scaling types, three linear (**scale**=GLINEARTYPE1-GLINEARTYPE3), one logarithmic (**scale**=GLOG10), and one discrete (**scale**=GDISCRETE). The different scaling types are shown below.

The three linear scaling types allow the user to specify either approximate or precise information for the number of intervals or the range of data. For **scale**=GLINEARTYPE1 the user provides approximate information and GINOGRAF will calculate new limits to include the range specified using sensible values for the interval between major tick marks. For **scale**=GLINEARTYPE2 the user provides approximate values for the range but a precise number of intervals. Again GINOGRAF will calculate new limits to include the range supplied. For **scale**=GLINEARTYPE3 the precise values for axis range and number of intervals is used.

Where logarithmic scaling is requested (**scale**=GLOG10), GINOGRAF will round up the axis range to the next power of 10 as shown below. The value of **nints** is not used. Note that neither axis ranges or data supplied to Graph or Chart routines can be less than or equal to zero if logarithmic axes are used.



Various axis scaling types

Discrete axes are used for Histograms and Bar Charts and for this type of axes **nints** sets the number of columns. The precise values of **vbeg** and **vend** are used to calculate the values displayed at the tick mark in the centre of the column. In terms of GINOGRAF, true graphs may only be drawn on the linear or logarithmic scales as they are continuous, however, graph drawing may be added to discrete axes (eg, a straight line fitted to a bar chart) without any adjustment (see mixing different graph types).

Each axis has a default scale type and range as follows:

```
scale = GLINEARTYPE3
nints = 9
vbeg  = 1.0
vend  = 10.0
```

These settings are unlikely to be satisfactory for any real data sets. Therefore of all the axis definition routines that may be used, ggSetAxesScaling() is the most necessary before drawing any graph form. The user should note also that these defaults are restored after calling the routine

ggRestoreAxesSettings().

Successive calls to ggSetAxesScaling() for a particular axis override previous calls for that axis. Once a position and scaling type has been defined, coordinates defined in terms of these axes are referred to as graphical axes coordinates.

Axes Position and Scaling Enquiry

The user may enquire the current position of either axis at any time using the routine ggEnqAxesPos():

ggEnqAxesPos(<u>or</u>,<u>xor</u>,<u>yor</u>,<u>axlen</u>,xory)

where **xory** determines which axis positions are being enquired (**xory**=GXAXIS for X axis and **xory**=GYAXIS for Y axis). **or**, **xor**,**yor** and **axlen** return the position and length in the same manner as supplied to ggSetAxesPos().

The default position for the X and Y axes is shown above (being the bottom and left axes drawn in the box). These defaults are restored after calling the routine ggRestoreAxesSettings().

As explained above, for certain types of scaling GINOGRAF will adjust the requested number of intervals and increase the requested range of data values in order to display an axis with sensible annotation. The user may enquire the actual settings that GINOGRAF will use when it draws the axes with the routine ggEnqAxesScaling():

ggEnqAxesScaling(<u>scale</u>,<u>nints</u>,<u>vbeg</u>,<u>vend</u>,xory)

where **xory** determines which axis scale parameters are being enquired (**xory**=GXAXIS for X axis and **xory** = GYAXIS for Y axis). **scale** returns the current type of scaling set for the axis; **nints** gives the actual number of intervals or columns on the axis and **vbeg** and **vend** give the actual data range of the axis.

Axis Sub-Intervals and Tick Mark Size

The number of sub-intervals (ie. minor tick marks between the major intervals) on an axis is controlled by the amount of space available when drawing the axis. This in turn is controlled by the current character size used for the annotation. The character height controls the number of sub-intervals on the Y axis (and the size of the tick marks on the X axis) and the character width controls the number of sub-intervals on the X axis (and the size of the tick marks on the Y axis).

Axes Drawing

The two routines available for optionally drawing the axes are:

`ggDrawAxes()`	for drawing the axes with or without numeric annotation and tick marks
`ggAddGrid()`	for drawing a complete four sided axis frame

Single Axes

Once an axis position and scaling type have been defined the axes may be drawn with the routine ggDrawAxes():

ggDrawAxes(tick,tickside,val,xory)

where **xory** determines which axis is required (**xory** = GXAXIS for X axis and **xory** = GYAXIS for Y axis).

tick, **tickside** and **val** determine whether the major and/or minor tick marks are to be drawn and whether values are drawn and on which side of the axis they are to appear. If the tick marks and values are required on the same side of the axis then **tickside** and **val** must be equal.

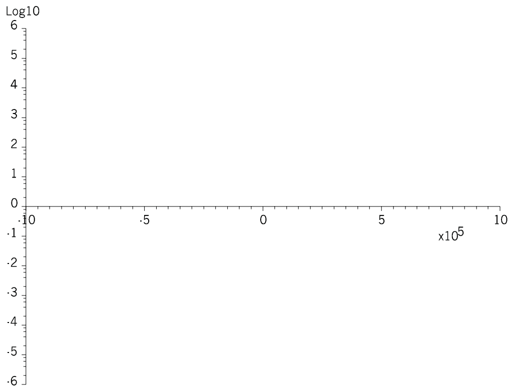
By default, the numeric annotation on a linearly scaled axis (scale≤3) is written in the form:

```
           N
 S = P * 10
```

where S is the true value of an axis tick mark, P is the actual number written by the axis tick mark with up to two decimal places and *10 to the power N is a multiplier written at the end of the axis, ensuring N is not in the range -2 to 2.

The numeric annotation on a logarithmically scaled axis is a set of consecutive integers. N represents log10. Intermediate tick marks may be drawn, depending on the available space. The scale factor LOG10 is written at the end of the axis.

An example of linear and logarithmic scaling is shown below.



Linear and logarithmic axes with default scaling

Further control over axis annotation position and format is provided by the annotation control routines ggSetAxesAnnotation() and ggSetAxesAttribs().

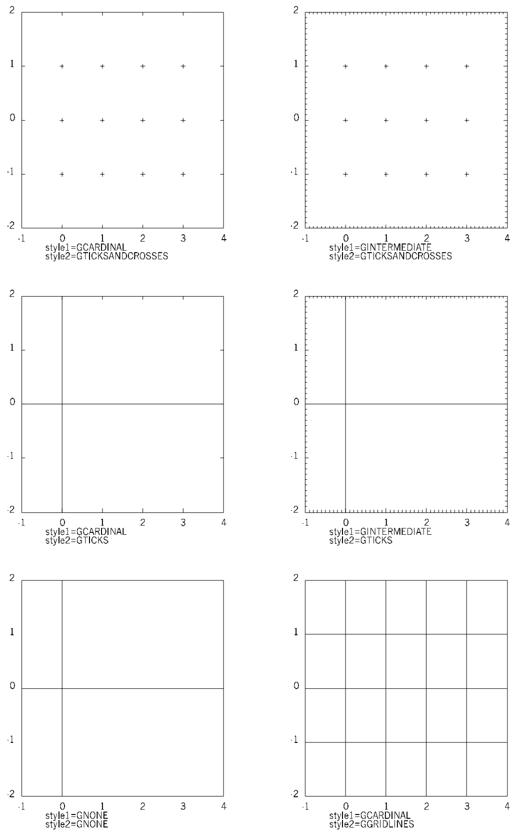
Axes Frames

A complete four sided frame may be drawn using the routine ggAddGrid():

ggAddGrid(style1,style2,anx,any)

where **style1** and **style2** determines the grid style as shown below. **anx** and **any** determine whether numeric annotation and/or grid lines or cross lines are drawn for either the X or Y axes. If drawn, tick marks are drawn on the inside of the frame and annotation is drawn on the outside of the frame. Grid positioning and scaling is controlled by the current settings of ggSetAxesPos() and ggSetAxesScaling().

Six of the possible values for **style1** and **style2** produce the following frame types:



Various axes frame types

As with ggDrawAxes(), further control over axis annotation position and format is provided by the annotation control routines ggSetAxesAnnotation() and ggSetAxesAttribs().

The default symbol drawn at the grid intersection points for styles using GTICKSANDCROSSES is a cross as shown above. This symbol may be changed to be any of the GINO symbols using the routine ggSetGridMarker():

```
ggSetGridMarker(sym)
```

where **sym** defines the symbol required. Any of GINO's standard, software or hardware symbols may be used as a grid intersection symbol, further details of which are found in the GINO documentation for gDrawMarker().

The current grid intersection symbol can be enquired through the routine ggEnqGridMarker():

```
ggEnqGridMarker(sym)
```

Numeric Annotation Format Control

The format of axes annotation can be altered with the routine ggSetAxesAnnotation():

```
ggSetAxesAnnotation(ndp,npower,asty, xory)
```

The first two arguments, **ndp** and **npower**, control the format of the numeric output, whereas **asty** sets the type of axis scale factor (in connection with **npower**). **xory** is a flag that determines whether the format parameters refer to the X or Y axis. If **xory** = GXAXIS, the format of the X axis is defined; if **xory** = GYAXIS, the format of the Y axis is defined. For log axes, **ndp** and **asty** are not used and the default annotation form is. (ie. where the value at a major tick mark is 10 to the power n, ‘n’ is displayed). **npower** can be used to define an output format at each major tick mark of either the actual values or values in the form 10 to the power n. Thus values less than 10 to the power **npower** are displayed as a real value and values greater than or equal to 10 to the power **npower** are displayed in exponential form.

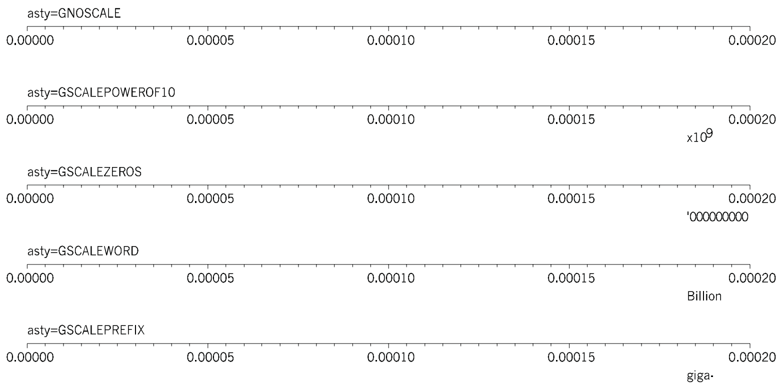
For non-log axes, the number of decimal places displayed for each value is determined using the parameter **ndp**. Positive values for **ndp** define the maximum number of decimal places that will be output if required, whereas negative values force that number of decimal places whether needed or not. Values may also be output scaled by a power of ten determined by the value of **npower** within the range -15 to 15; if **npower** is outside this range then GINOGRAF calculates a suitable value.

asty sets the type of display for the axis scale factor when drawing axes, offering a number of engineering and scientific forms for example:

asty	**for 10 to the power 3**	**for 10 to the power -3**
GNOSCALE	-	-
GSCALEPOWEROF10	10^{3}	10^{-3}
GSCALEZEROS	‘000	0.00’
GSCALEWORD	Thousand	Thousandths
GSCALEPREFIX	kilo-	milli-

N.B. for **asty**=GNOSCALE, no scale factor is displayed even though the values displayed are divided by 10 to the power **npower**. For types GSCALEWORD or GSCALEPREFIX, the scale factor **npower** is required to be a multiple of 3.

An example of the five types is shown below where **ndp**=-5 and **npower**=9.



The five axes scaling types

ggSetAxesAnnotation() is also used to control the numeric format of other values output by GINOGRAF. These include the output of Graph and Chart data values from the ggAddxxxValues() routines and values output by the Text Chart routines. In most cases the values will be associated with a particular axes and so they will match those output on the axes itself, but in other cases (Pie Charts and Text Charts) the format of the Y axis is used. Users should also note that all the data values output will be scaled by 10 to the power **npower** and therefore **npower** should only be set to non-zero values where axes are displayed (and hence the relevant scale factor) or where the user displays the scale factor manually.

Where a power factor is used on an axis and non scaled data values are required for the Graph or Chart annotation, ggSetAxesAnnotation() must be called with **npower** set to zero before the data values are output, eg,

C Code

```
/* SET REQUIRED FORMAT FOR AXIS */
   ggSetAxesAnnotation(-2,3,GSCALEZEROS,GXAXIS);
   ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS);
/* SWITCH OFF SCALE FACTOR */
   ggSetAxesAnnotation(-2,0,GNOSCALE,GXAXIS);
   ggAddxxxValues(   );
```

F90 Code

```
! SET REQUIRED FORMAT FOR AXIS
  call ggSetAxesAnnotation(-2,3,GSCALEZEROS,GXAXIS)
  call ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS)
! SWITCH OFF SCALE FACTOR
  call ggSetAxesAnnotation(-2,0,GNOSCALE,GXAXIS)
  call ggAddxxxValues(   )
```

The current attributes for numeric and axis format control for each axis are returned through ggEnqAxesAnnotation():

ggEnqAxesAnnotation(<u>ndp</u>,<u>npower</u>,<u>nrfigs</u>,<u>asty</u>,xory)

where **xory** is supplied to specify the required axis. The remaining arguments are as set by the routine ggSetAxesAnnotation() except **nrfigs** which is an additional argument giving the total field width of the annotation on the specified axis.

Axes Titling

The routine ggDrawAxesTitle() provides a means to output a justified title with reference to either the X or Y axis:

ggDrawAxesTitle(string,yorx,xory,pos1,pos2)

where **string** is a character variable or constant holding the title. **xory** determines the axis to which the title is related whereas **pos1** & **pos2** determines the vertical and horizontal justification within the limits of the current position and length of the specified axis.

For titles related to the X axis (**xory** = GXAXIS), **yorx** is the Y coordinate position in user space coordinates. Conversely for Y axis titles (**xory** = GYAXIS), **yorx** is the X coordinate position in user space coordinates. X axis titles are written to be read from left to right, and Y axis titles are written with the first character lowest.

As ggDrawAxesTitle() uses a user space coordinate to determine the position of the title, its use is not restricted to axis titling, but may be used for more general titling such as at the top of a graph or even to label a point of interest within a graph. Where the position needs to be related to some point on the graphical coordinate system it may be necessary to convert between a graphical and space coordinates system in order to supply the value of **yorx**. This can be achieved with the utility routine ggTransformGraphPoint().

An example of the use of ggDrawAxesTitle() is shown below, displaying the interaction of **yorx**, **xory**, **pos1** & **pos2** with reference to the X axis.

Axes Labelling

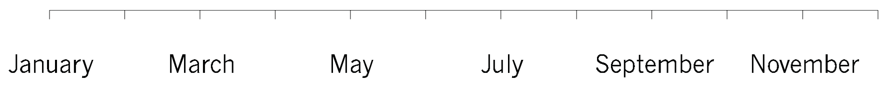
The routine to independently label an axis is ggDrawAxesLabels():

ggDrawAxesLabels(nstr,string,iv,xory)

where **string** is a character array of dimension **nstr**, containing labels to be output at major tick marks and **xory** determines which axes is to be labelled. The labels in **string** are cycled if there are greater than **nstr** tick marks on the axis. The parameter **iv** determines whether the labels are output on the anticlockwise (**iv**=GANTICLOCKWISE) or clockwise (**iv**=GCLOCKWISE) side of the axis.

Although ggDrawAxesLabels() provides the means to position the labels at any position, in general these axes labelling routines are used as an alternative to numeric labelling. Therefore the axis to which labelling is to be added should be drawn without numeric labelling by setting the appropriate argument, ie, **anx**=0 or **any**=0 in ggAddGrid(), or **val**=0 in ggDrawAxes() if required.

The example below shows the use of ggDrawAxesLabels(). Note that, by default, some of the labels are suppressed because the length is too long to fit between tick marks. Further control of label output is given through the routine ggSetAxesAttribs().



Axis labelling using ggDrawAxesLabels

C Code

```
/*  AXIS LABELLING WITH ggDrawAxesLabels() */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GDIM  paper;
  int   papty;
  char *mons[12]= { “January”,"February","March",
    ”April”,"May","June","July","August","September",
    ”October”,"November","December" };


  gOpenGino();
  xxxxx();
  gEnqDrawingLimits(&paper,&papty);
  ggSetGraphCharMode(GGINOMODE);
/* DEFINE AXIS POSITION */
  ggSetAxesPos(GDATAORIGIN,0.2*paper.xpap,
    0.2*paper.ypap,0.7*paper.xpap,GXAXIS);
/* DEFINE AXIS RANGE */
  ggSetAxesScaling(GLINEARTYPE3,11,1.0,12.0,GXAXIS);

/* DRAW AXIS WITHOUT ANNOTATION */
  ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS);
/* LABEL AXIS */
  ggDrawAxesLabels(12,mons,GCLOCKWISE,GXAXIS);

  gSuspendDevice();
  gCloseGino();
  return(0);
}
```

F90 Code

```
!  AXIS LABELLING WITH ggDrawAxesLabels()
use gino_f90
use graf_f90


  type (GDIM) paper
  integer papty
  character (len=9), dimension(12) :: mons = &
  (/'January','February','March','April','May', &
    'June','July','August','September','October', &
    'November','December'/)

  call gOpenGino
  call xxxxx
  call gEnqDrawingLimits(paper,papty)
  call ggSetGraphCharMode(GGINOMODE)

! DEFINE AXIS POSITION
  call ggSetAxesPos(GDATAORIGIN,0.2*paper%xpap, &
    0.2*paper%ypap,0.7*paper%xpap,GXAXIS)
! DEFINE AXIS RANGE
  call ggSetAxesScaling(GLINEARTYPE3,11,1.0,12.0,GXAXIS)

! DRAW AXIS WITHOUT ANNOTATION
  call ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS)
! LABEL AXIS
  call ggDrawAxesLabels(12,mons,GCLOCKWISE,GXAXIS)

  call gSuspendDevice
  call gCloseGino
  stop
  end
```

Axes Annotation Control

ggSetAxesAttribs() controls both numeric and textual annotation output by the complete graph and chart routines and the routines ggAddGrid(), ggDrawAxes(), and ggDrawAxesLabels():

ggSetAxesAttribs(swi,xy,nstart,nskip,aoff,angstr,jusver,jushor,reduc,xory)

It controls the following attributes:

- Position with respect to the axes (**swi**,**xory**)
- Start tick mark (**nstart**)
- Number of tick marks to skip (**nskip**)
- Offset from control point (**aoff**)
- String angle of annotation (**angstr**)
- Justification of string or value (**jusver**,**jushor**)
- Optional text size reduction (**reduc**)
- with **xory** determining the X or Y axis.

Annotation Position

Annotation can be positioned at two places with respect to the axes. The default position (**swi**=GONAXIS) is at a fixed distance either side of the axis alongside the major tick marks. The second (**swi**=GOFFSET) is either side of a user defined position in user space coordinates specified by the argument **xory**. This second option is useful for either adding additional annotation or moving the annotation out of the graph area where axes have been placed in the centre.

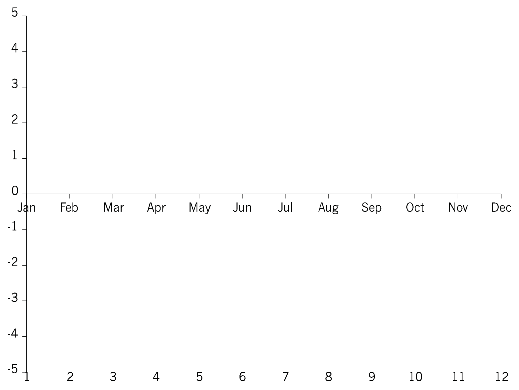
C Code

```
/*  ANNOTATION POSITIONS SET BY ggSetAxesAttribs() */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GDIM  paper;
  int   papty;
  char *mons[12] = { “Jan”,"Feb","Mar","Apr","May",
    ”Jun”,"Jul","Aug","Sep","Oct","Nov","Dec" };

  gOpenGino();
  xxxxx();
  gEnqDrawingLimits(&paper, &papty);
  ggSetGraphCharMode(GGINOMODE);

/* DEFINE AXIS POSITIONS */
  ggSetAxesPos(GAXISSTART,0.1*paper.xpap,
    0.5*paper.ypap,0.8*paper.xpap,GXAXIS);
  ggSetAxesPos(GDATAORIGIN,0.1*paper.xpap,
    0.5*paper.ypap,0.8*paper.ypap,GYAXIS);
/* DEFINE AXIS RANGES */
  ggSetAxesScaling(GLINEARTYPE3,11,1.0,12.0,GXAXIS);
  ggSetAxesScaling(GLINEARTYPE3,10,-5.0,5.0,GYAXIS);

/* DRAW Y AXIS WITH ANNOTATION */
  ggDrawAxes(GCARDINAL,GANTICLOCKWISE,
    GANTICLOCKWISE,GYAXIS);

/* LABEL X AXIS WITH MONTHS AT DEFAULT POSITION */
  ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS);

/* DRAW X AXIS WITH VALUES AT BOTTOM OF Y AXIS */
  ggSetAxesAttribs(GOFFSET,0.1*paper.ypap,1,0,0.0,
    0.0,GDEFAULTPOSITION,GDEFAULTPOSITION,GNOREDUCE,
    GXAXIS);
  ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS);

  gSuspendDevice();
  gCloseGino();
  return(0);
}
```

F90 Code

```
!  ANNOTATION POSITIONS SET BY
!           ggSetAxesAttribs()
use gino_f90
use graf_f90


  type (GDIM) paper
  character (len=3), dimension(12) :: mons = &
    (/'Jan','Feb','Mar','Apr','May','Jun',
      'Jul','Aug','Sep','Oct','Nov','Dec'/)

  call gOpenGino
  call xxxxx
  call gEnqDrawingLimits(paper, ipapty)
  call ggSetGraphCharMode(GGINOMODE)

! DEFINE AXIS POSITIONS
  call ggSetAxesPos(GAXISSTART,0.1*paper%xpap, &
    0.5*paper%ypap,0.8*paper%xpap,GXAXIS)
  call ggSetAxesPos(GDATAORIGIN,0.1*paper%xpap, &
    0.5*paper%ypap,0.8*paper%ypap,GYAXIS)
```

```
        ! DEFINE AXIS RANGES
          call ggSetAxesScaling(GLINEARTYPE3,11,1.0,12.0,GXAXIS)
          call ggSetAxesScaling(GLINEARTYPE3,10,-5.0,5.0,GYAXIS)
        ! DRAW Y AXIS WITH ANNOTATION
          call ggDrawAxes(GCARDINAL,GANTICLOCKWISE, &
            GANTICLOCKWISE,GYAXIS)

        ! LABEL X AXIS WITH MONTHS AT DEFAULT POSITION
          call ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS)

        ! DRAW X AXIS WITH VALUES AT BOTTOM OF Y AXIS
          call ggSetAxesAttribs(GOFFSET,0.1*paper%ypap,1,0, &
            0.0,0.0,GDEFAULTPOSITION,GDEFAULTPOSITION,GNOREDUCE, &
            GXAXIS)
          call ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```



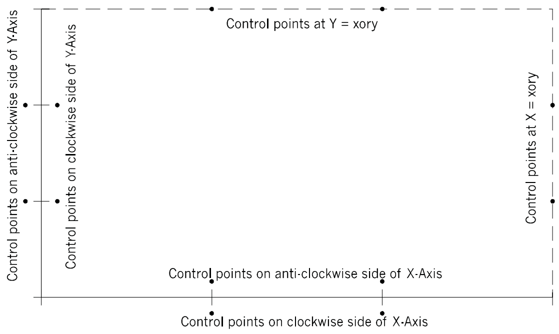
Annotation position set by ggSetAxesAttribs

Skipping labels

By default, as in the above example, all the axes labels, whether numeric or textual are output, from the first to the last tick mark. There may be conditions where labels need to be omitted, either for clarity or because they are too long. ggSetAxesAttribs() can be used to control which tick mark is labelled first and whether labels are to be skipped. The arguments **nstart** and **nskip** are used for this purpose. Thus if **nstart**=2 and **nskip**=1 only the even numbered tick marks would be labelled. The default condition exists when **nskip** is less than 0. Here **nskip** is automatically calculated to skip sufficient labels so that they do not overlap each other. If **nskip**=GNONE all the annotation values are output, which may result in overlapping characters if there is insufficient space (see use of **reduc** below).

Adjusting Offsets, Angle and Justification

An offset, the string angle and justification of each label can also be adjusted with ggSetAxesAttribs(). In order to appreciate this facility it is necessary to know the position of the control points about which the adjustments are made. Where the annotation is made on the axis itself (**swi**=GONAXIS) the control points are twice the tick mark length away from the axis, either on the clockwise or anti-clockwise side depending on the choice made in ggDrawAxes() or ggDrawAxesLabels(). Where the annotation is at the defined position **xory** (**swi**=GOFFSET) the control points are at the coordinate **xory** in line with the major tick marks on the corresponding axis. The control points are shown below.


Control points at Y = xory
Control points on anti-clockwise side of Y-Axis
Control points on clockwise side of Y-Axis
Control points at X = xory
Control points on anti-clockwise side of X-Axis
Control points on clockwise side of X-Axis

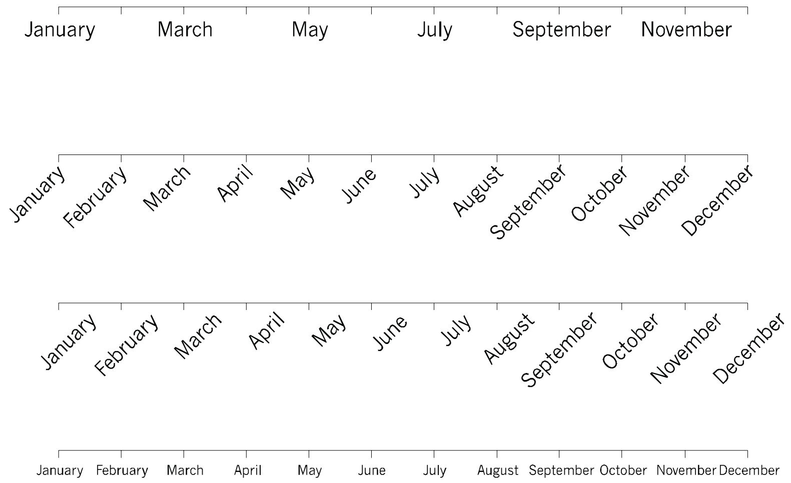
Axes annotation control points

By default the offset, string angle and justifications are as shown in the following table

Control Point	Offset	Angle	Vertical Justification	Horizontal Justification
Clockwise side of X axis or Y=xory	0.0	0.0	Top	Centre
Anti-clockwise side of X axis or Y=xory	0.0	0.0	Bottom	Centre
Clockwise side of Y axis or X=xory	0.0	0.0	Bottom	Left
Anti-clockwise side of Y axis or X=xory	0.0	0.0	Bottom	Right

Each of these settings can be changed for either the X axis or Y axis with the arguments **aoff**, **angstr**, **jusver**, **jushor** respectively. The offset (**aoff**) is measured as a proportion of the distance between major tick marks on the specified axis. ie. where **aoff**=0.5, the annotation is drawn midway between the major tick mark to which it refers and the next major tick mark as shown in the example below. Altering the string angle of axes annotation is a useful method of fitting in long labels to each major tick mark on the X axis.

The final control offered by ggSetAxesAttribs() is an option to automatically reduce the size of the annotation to ensure it fits within the major tick marks without overlapping. The option is particularly useful where horizontal annotation is preferred and all the labels need to be output. When **reduc** is set to GREDUCE and **nskip**=0, the character size is reduced by the required amount (equally in both directions) so that the longest label on the axis fits between the specified number of annotated tick marks. An example showing the usage of axes annotation attributes is shown below.



Axes annotation control

C Code

```
/*  AXIS ANNOTATION CONTROL */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GDIM  paper;
  int   papty;
  char *mons[12] = { “January”,"February","March",
    ”April”,"May","June","July","August","September",
    ”October”,"November","December" };

  gOpenGino();
  xxxxx();
  gEnqDrawingLimits(&paper,&papty);
  ggSetGraphCharMode(GGINOMODE);

/* DEFINE AXIS RANGE */
  ggSetAxesScaling(GLINEAR3,11,1.0,12.0,GXAXIS);
/* DEFINE FIRST AXIS WITH DEFAULT ATTRIBUTES */
  ggSetAxesPos(GAXISSTART,0.2*paper.xpap,
    0.8*paper.ypap,0.7*paper.xpap,GXAXIS);
  ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS);
  ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS);

/* DEFINE SECOND AXIS WITH ANGLED TEXT */
  ggSetAxesPos(GAXISSTART,0.2*paper.xpap,
    0.6*paper.ypap,0.7*paper.xpap,GXAXIS);
  ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS);
  ggSetAxesAttribs(GONAXIS,0.0,1,GNONE,0.0,45.0,
    GMIDDLE,GRIGHT,GNOREDUCE,GXAXIS);
  ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS);

/* DEFINE THIRD AXIS WITH OFFSET ANGLED TEXT */
  ggSetAxesPos(GAXISSTART,0.2*paper.xpap,
    0.4*paper.ypap,0.7*paper.xpap,GXAXIS);
  ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS);
  ggSetAxesAttribs(GONAXIS,0.0,1,GNONE,0.5,45.0,
    GMIDDLE,GRIGHT,GNOREDUCE,GXAXIS);
  ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS);

/* DEFINE FOURTH AXIS WITH REDUCED TEXT */
  ggSetAxesPos(GAXISSTART,0.2*paper.xpap,
    0.2*paper.ypap,0.7*paper.xpap,GXAXIS);
  ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS);
  ggSetAxesAttribs(GONAXIS,0.0,1,GNONE,0.0,0.0,
    GDEFAULTPOSITION,GDEFAULTPOSITION,GREDUCE,GXAXIS);
  ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS);

  gSuspendDevice();
  gCloseGino();
  return(0);
}
```

F90 Code

```
!  AXIS ANNOTATION CONTROL
use gino_f90
use graf_f90


  type (GDIM) paper
  character (len=9), dimension(12) :: mons = &
    (/'January','February','March','April','May','June', &
      'July','August','September','October', 'November', &
      'December'/)

  call gOpenGino
  call xxxxx
  call gEnqDrawingLimits(paper, ipapty)
  call ggSetGraphCharMode(GGINOMODE)

! DEFINE AXIS RANGE
  call ggSetAxesScaling(GLINEAR3,11,1.0,12.0,GXAXIS)

! DEFINE FIRST AXIS WITH DEFAULT ATTRIBUTES
  call ggSetAxesPos(GAXISSTART,0.2*paper%xpap, &
    0.8*paper%ypap,0.7*paper%xpap,GXAXIS)
  call ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS)
  call ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS)
```

```
        ! DEFINE SECOND AXIS WITH ANGLED TEXT
          call ggSetAxesPos(GAXISSTART,0.2*paper%xpap, &
            0.6*paper%ypap,0.7*paper%xpap,GXAXIS)
          call ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS)
          call ggSetAxesAttribs(GONAXIS,0.0,1,GNONE,0.0,45.0, &
            GMIDDLE,GRIGHT,GNOREDUCE,GXAXIS)
          call ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS)

        ! DEFINE THIRD AXIS WITH OFFSET ANGLED TEXT
          call ggSetAxesPos(GAXISSTART,0.2*paper%xpap, &
            0.4*paper%ypap,0.7*paper%xpap,GXAXIS)
          call ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS)
          call ggSetAxesAttribs(GONAXIS,0.0,1,GNONE,0.5,45.0, &
            GMIDDLE,GRIGHT,GNOREDUCE,GXAXIS)
          call ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS)

        ! DEFINE FOURTH AXIS WITH REDUCED TEXT
          call ggSetAxesPos(GAXISSTART,0.2*paper%xpap, &
            0.2*paper%ypap,0.7*paper%xpap,GXAXIS)
          call ggDrawAxes(GCARDINAL,GCLOCKWISE,GNOVAL,GXAXIS)
          call ggSetAxesAttribs(GONAXIS,0.0,1,GNONE,0.0,0.0, &
            GDEFAULTPOSITION,GDEFAULTPOSITION,GREDUCE,GXAXIS)
          call ggDrawAxesLabels(12,mons,GANTICLOCKWISE,GXAXIS)
        !
          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Axes Annotation Enquiry

The user may obtain the parameters used for annotation control set up by the routine ggSetAxesAttribs() by calling ggEnqAxesAttribs():

ggEnqAxesAttribs(switch,xy,nstart,nskip,aoff,angstr,jusver,jushor,
reduc,xory)

The only parameter that the user needs to supply is **xory**, the number of the axis being enquired. All other parameters are returned.

Date Axes

In addition to the standard numerical axes provided by GINOGRAF, the library also provides a means of handling date data and displaying date axes of various formats. In order to integrate date data into the graph displaying features of GINOGRAF, dates are converted onto a numeric scale where the start of the Gregorian calendar (September 9th 1752) is equivalent to 1.0.

Dates are input as a character string of up to 10 characters, the delimiter between days, months and years is the forward oblique stroke “/”. Days and months can be given as one or two digits, with an optional leading zero for values less than 10. Years are given as two or four digits. If only two digits are given, they are assumed to apply to the range 1950 to 2049. ie. 05 is treated as 2005 and not 1905. Leading and trailing spaces are ignored. Examples of valid date forms are:

```
31/12/95
01/03/96
 1/ 3/96
1/3/96
29/2/00
29/2/2000
28/02/1900
```

The default ordering of day, month, year is the British form, but the American and reverse forms can be used by setting the date input format type using the routine ggSetDateFormat():

ggSetDateFormat(inform,insep,ouform,ousep)

where **inform** and **ouform** are either GBRITISH, GAMERICAN or GLOGICAL.

Date Axes Scaling

The user can define either the X(horizontal) or Y(vertical) axis of a graph to be a date axis using the routine ggSetDateAxesScaling():

ggSetDateAxesScaling(scale,dincr,dbeg,dend,xory)

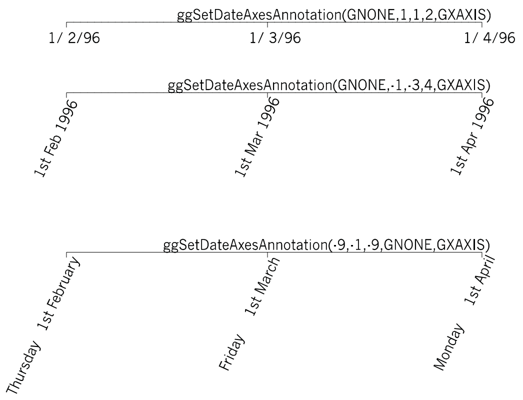
Three scale types are permitted to cater for different end conditions required by the user in conjunction with the specified increment type (as set in **dincr**). Unlike the numerical axes, the user specifies the type of increment required for the date axes in terms of decade, year, month etc., rather than the number of increments. The start and end points of the date axes are defined in the character strings **dbeg** and **dend** as described above.

As with numerical axes, date axes are positioned using the routine ggSetAxesPos() and displayed using the axes drawing routine ggDrawAxes(). They are also divided into a number of increments separated by major tick marks, which for the date axes represents the specified increment type selected by ggSetDateAxesScaling(). The axes may include minor tick marks representing the next smaller increment type according to the argument of ggDrawAxes(). Labels are (optionally) drawn at each major tick mark according to the current date output format as defined by the routine ggSetDateAxesAnnotation():

```
ggSetDateAxesAnnotation(fdow,fday, fmon,fyear,xory)
```

where the first four arguments specify the requirement and/or format of each component of the date to be output. Each of the ‘day of the week’, day or months may be output alphanumerically or numerically and different output formats may be specified for each of the X or Y axes.

Examples of some of the different output formats are shown below.


ggSetDateAxesAnnotation(GNONE,1,1,2,GXAXIS)
1/ 2/96
1/ 3/96
1/ 4/96
ggSetDateAxesAnnotation(GNONE,-1,-3,4,GXAXIS)
1st Feb 1996
1st Mar 1996
1st Apr 1996
ggSetDateAxesAnnotation(-9,-1,-9,GNONE,GXAXIS)
Thursday 1st February
Friday 1st March
Monday 1st April

Date axes output formats

Date Conversion

Where either of the X or Y axis has been defined as a date axis using the routine ggSetDateAxesScaling(), GINOGRAF defines an internal numerical scaling for the range of dates used for the axes based on the Gregorian calendar as described above.

It is necessary therefore to convert 'date' data to this numerical scale so that the various graph and chart output forms described later in this manual can be used with the date axes. A single routine is provided for this purpose, to convert an array of date data into its equivalent numeric values.

ggConvertDates(ndates,dates,data)

where **dates** is a character array containing **ndates** dates in the current date input format, returning the numerical values in **data**.

Single dates may be converted to and from their numerical values using the following routines:

ggConvertDateToGraph(date,value)

ggConvertGraphToDate(value,date)

Date Axes Usage

Two examples show the usage of date axes:
Example 1 - The user has an array of dates and numeric values which need to be plotted on date axes.

The following input data represents maximum and minimum temperature readings:

```
01/01/96    3    -1
02/01/96    4    -1
03/01/96    5    0
04/01/96    2    -3
05/01/96    0    -6
:
```

The following code segment reads in the data, sets up the date axes and converts the data ready for graph drawing:

C Code

```
#define MAX_DATA 100
  char  *Dates[MAX_DATA];
  float X[MAX_DATA];
  FILE  *ifp;
  int   n, stat, Max[MAX_DATA],Min[MAX_DATA];

/* Allocate space for dates MAX_DATA times */
  for (n=0; n<MAX_DATA; n++)
    Dates[n]=(char*)malloc(9*sizeof(char));
```

```
        /* READ IN TEMPERATURE DATA */
          if (!(ifp = fopen("temps.dat","r"))) return 1;
          n=0;
          stat=fscanf(ifp,"%s%d%d", Dates[n],&Max[n],&Min[n]);
          while (stat!=EOF) {
            n++;
            stat=fscanf(ifp,"%s%d%d\n",Dates[n],&Max[n],&Min[n]);
          }
          fclose(ifp);

        /* DEFINE DATE AXES */
          ggSetDateAxesScaling(1,GDAY,Dates[0],Dates[n-1],GXAXIS);

        /* DRAW DATE AXES */
          ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS);

        /* CONVERT DATE DATA FOR PLOTTING AGAINST X AXIS */
          ggConvertDates(n,Dates,X);
```

F90 Code

```
         integer, parameter :: MAX_DATA=100
           character (len=8), dimension(MAX_DATA) :: Dates
           real X(MAX_DATA)
           integer n,Max(MAX_DATA),Min(MAX_DATA)

        ! READ IN TEMPERATURE DATA
           open(unit=11,file='TEMPS.DAT')
           n=0
        10 read(11,11,end=20) Dates(n+1),Max(n+1),Min(n+1)
        11 format(A,2I4)
           n=n+1
           goto 10

        ! DEFINE DATE AXES
        20 call ggSetDateAxesScaling(1,GDAY,Dates(1),&
             Dates(n),GXAXIS)

        ! DRAW DATE AXES
           call ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS)

        ! CONVERT DATE DATA FOR PLOTTING AGAINST X AXIS
           call ggConvertDates(n,Dates,X)
```

Example 2 - The user has numeric data which actually represents date data and needs to be displayed on date axes. This can be handled in one of two ways; either all the data can be handled numerically superimposing a date axis over the numeric axis, or you can convert the numeric data to the required date scale. For example, if the following data represents 1996 monthly shoe sales:

```
1    223
2    154
3    149
4    152
5    123
6    152
7    278
8    121
9    119
10   125
11   98
12   129
```

The following code segment sets up and draws a date axis and superimposes a numeric axis with the same number of intervals over it to do the plotting of the graph:

C Code

```
   int month[12],sales[12];
   FILE *ifp;
   int n, stat;

/* READ IN SHOE SALES DATA */
   if (!(ifp=fopen(“shoe.dat”,"r"))) return 1;
   n=0;
   stat=fscanf(ifp,"%d%d", &month[n],&sales[n]);
   while (stat!=EOF) {
     n++;
     stat=fscanf(ifp,"%d%d",&month[n],&sales[n]);
   }
   fclose(ifp);

/* DEFINE AND DRAW DATE AXES FOR SALES PERIOD */
   ggSetDateAxesScaling(3,GMONTH,”01/01/96”,”01/12/96”,GXAXIS);
   ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS);

/* RE-DEFINE NUMERIC SCALING FOR SALES PERIOD */
   ggSetAxesScaling(GLINEARTYPE3,11,1.0,12.0,GXAXIS);

/* PLOT DATA ACCORDING TO NUMERIC SCALING */
```

F90 Code

```
  integer month(12),sales(12)
  integer n

!  READ IN SHOE SALES DATA
   open(unit=11,file=’SHOE.DAT’)
   n=0
10 read(11,*,end=20) month(n+1),sales(n+1)
   n=n+1
   goto 10
```

```
!  DEFINE AND DRAW DATE AXES FOR SALES PERIOD
20 call ggSetDateAxesScaling(3,GMONTH,’01/01/96’,‘01/12/96’,GXAXIS)
   call ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS)

!  RE-DEFINE NUMERIC SCALING FOR SALES PERIOD
   call ggSetAxesScaling(GLINEARTYPE3,11,1.0,12.0,GXAXIS)

!  PLOT DATA ACCORDING TO NUMERIC SCALING
```

And the following code would use date axes and convert the data appropriately:

C Code

```
   int month[100],sales[100];
   char *mon[100];
   int n, i, stat, m1, m2;

/* Allocate space for dates */
   for (n=0; n<100; n++)
      Mons[n]=(char*)malloc(9*sizeof(char));

/* READ IN SHOE SALES DATA */
   if (!(ifp=fopen(“shoe.dat”,"r"))) return 1;
   n=0;
   stat=fscanf(ifp,"%d%d",&month[n],&sales[n]);
   while (stat!=EOF) {
     n++;
     stat=fscanf(ifp,"%d%d",&month[n],&sales[n]);
   }
   fclose(ifp);

/* CONVERT MONTH NO. TO DATE SCALING */
   for (i=0;i<n; i++) {
     m1=(int)(month(i))/10;
     m2=(int)(month(I))%10;
     sprintf(mon(i), “01/%d%d/96", m1+48, m2+48);
   }
   ggConvertDates(n,mon,month);

/* DEFINE AND DRAW DATE AXES FOR SALES PERIOD */
   ggSetDateAxesScaling(3,GMONTH,”01/01/96”,”01/12/96”,GXAXIS);
   ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS);

/* PLOT DATA ACCORDING TO DATE AXES SCALING */
```

F90 Code

```
    integer month(100),sales(100)
    character (len=8), dimension(100) :: mon
    integer n, i

! READ IN SHOE SALES DATA
    open(unit=11,file=’SHOE.DAT’)
    n=0
10  read(11,*,end=20) month(n+1),sales(n+1)
    n=n+1
    goto 10
```

```
         ! CONVERT MONTH NO. TO DATE SCALING
         20  DO 30 I=1,N
                m1=int(month(i))/10
                m2=mod(int(month(i)),10)
                mon(i)='01/' // char(m1+48) // char(m2+48)// '/96'
         30  continue
             call ggConvertDates(mon,month,n)

         ! DEFINE AND DRAW DATE AXES FOR SALES PERIOD
             call ggSetDateAxesScaling(3,GMONTH,'01/01/96','01/12/96',GXAXIS)
             call ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS)

         ! PLOT DATA ACCORDING TO DATE AXES SCALING
```

Both programs produce exactly the same output.

Note that GINOGRAF always draws axes with equally spaced tickmarks and therefore date intervals of ‘month’ or ‘year’ are not drawn at exactly the correct ‘daily’ position because of the unequal number of days in each of those intervals.

Date Axes Enquiry

The current date axis settings can be enquired though the following three routines:

ggEnqDateFormat(<u>inform</u>,<u>insep</u>,<u>ouform</u>,<u>ousep</u>)

ggEnqDateAxesScaling(<u>scale</u>,<u>dincr</u>,<u>dbeg</u>,<u>dend</u>,xory)

ggEnqDateAxesAnnotation(<u>fdow</u>,<u>fday</u>,<u>fmon</u>,<u>fyear</u>,xory)

Default Restoration

Axes parameters may be restored to their numeric defaults using the routine:

ggRestoreAxesSettings()

which resets the conditions set up using ggSetAxesPos(), ggSetAxesScaling() and ggSetAxesAttribs() for both axes.

If some of the axes setup is to be retained, or if only one axis’ settings is to be reset then the relevant enquiry routines must be called in order to save the appropriate settings; these conditions may be set up again using ggSetAxesPos(), ggSetAxesScaling() and ggSetAxesAttribs() after a call to ggRestoreAxesSettings().

Chapter 3

GRAPHS

Introduction to Graphs

This chapter concerns itself with the production of Graphs. The GINOGRAF definition of a graph is a two dimensional representation of coordinate data on continuous axes. The graph displaying routines are split into two main groups: Complete Graph Drawing routines which produce graphs with a single routine call; and lower level component routines which build up a graph in a modular fashion.

Summary of the Graph facilities available:

`ggPlotGraph()`	Draws single data set on set of axes with automatic scaling
`ggSetGraphScaling()`	Controls ggPlotGraph() data ranges and axes intervals
`ggAddGraphPolyline()`	Draws polyline graph on current axes
`ggAddGraphCurve()`	Draws curved graph on current axes
`ggAddAkimaCurve()`	Draws Akima curved graph on current axes
`ggAddGraphSpline()`	Draws spline curve graph on current axes
`ggSetCurveStartConds()`	Sets start conditions of curve graph
`ggSetCurveEndConds()`	Sets end conditions of curve graph
`ggAddGraphMarkers()`	Draws set of symbols on current axes
`ggAddErrorBars()`	Draws error bar graph on current axes
`ggAddSquareWave()`	Draws square wave graph on current axes
`ggAddPopulationGraph()`	Draws a population graph on current axes
`ggFillBelowDataset()`	Fills area below polyline
`ggFillBetweenDatasets()`	Fills area between two data sets
`ggAddGraphValues()`	Annotates data points on current axes
`ggSetValueAttribs()`	Sets data value display attributes
`ggSetValueTags()`	Sets prefix and suffix strings for data value output

Complete Graph Drawing

The routine described in this section is complete in itself. The user simply provides a set of data in an array of type GPOINT and makes a single routine call. The position and scaling of the graph is calculated automatically and output is drawn to fit the current graph drawing area (see page 16). The complete graph drawing routine provide by GINOGRAF is ggPlotGraph():

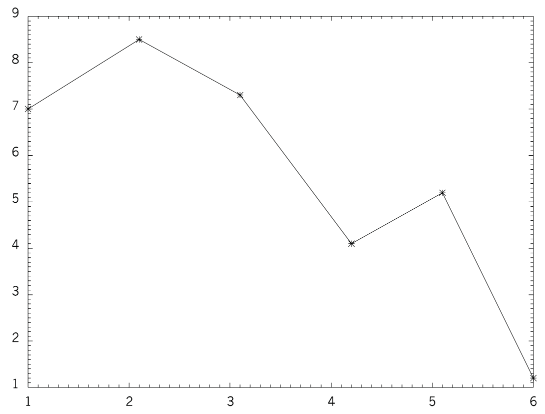
ggPlotGraph(npts,points,scx,scy,style,axis)

where **npts** of data are supplied in the array **points**. Axes can be represented by a frame (**axis**=GFRAME), or by two axes intersecting at the data origin (if present) or at the bottom left of the graph (**axis**=GAXIS). Each may be linear or logarithmically scaled depending on the value of **scx** and **scy** (GLINEAR = Linear scale on axes, GLOG10 = Logarithmic scale on axes). The data set may be represented in various ways depending on the value of **style** as follows:

style	**Lines**	**Marker**
`-GAKIMA`	Akima Curve	Asterisk
`-GSPLINE`	Spline Curve	Asterisk
`-GCUBIC`	Curve	Asterisk
`-GSTRAIGHT`	Straight Line	Asterisk
`GSYMBOLS`	None	Asterisk
`GSTRAIGHT`	Straight Line	None
`GCUBIC`	Curve	None
`GSPLINE`	Spline Curve	None
`GAKIMA`	Akima Curve	None

The axes are drawn to include at least the complete range of values in the **points** array, possibly rounding up the limits in order to use reasonable intervals. The numerical annotation on each axis is subject to the current settings of format and annotation control set by ggSetAxesAnnotation() and ggSetAxesAttribs().

An example of output produced by ggPlotGraph() is shown below.



Example output from ggPlotGraph routine

The following code shows the program that generated the figure above followed by an equivalent program using the low-level component routines described in the next chapters. While axis and graph titles can be added after the routine ggPlotGraph() has been called, the full flexibility of layout and style can only be achieved using these component routines as the routine ggPlotGraph() is provided to present user data as quickly as possible with the minimum of effort.

C Code

```
/*  USE OF COMPLETE GRAPH DRAWING ROUTINE
   ggPlotGraph() */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GPOINT pnts[6] = {1.0, 7.0,2.1, 8.5,3.1, 7.3,
                    4.2, 4.1,5.1, 5.2,6.0, 1.2};

  gOpenGino();
  xxxxx();
  gNewDrawing();
  ggSetGraphCharMode(GGINOMODE);

  ggPlotGraph(6,pnts,GLINEAR,GLINEAR,-GSTRAIGHT,GFRAME)

  gSuspendDevice();
  gCloseGino();
  return 0;
}
```

```
/*  USING LOW LEVEL COMPONENT ROUTINES */
#include <gino-c.h>
#include graf-c.h>

int main(void) {
   GPOINT pnts[6] = {1.0, 7.0,2.1, 8.5,3.1, 7.3,
                     4.2, 4.1,5.1, 5.2,6.0, 1.2};

   int flg;
   GLIMIT lims;
   GCHASTY rep;

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
   ggEnqPlotFrame(&flg,&lims);
   gEnqCharAttribs(&rep);

/* SET UP AXES POSITIONS AND SCALES */
   ggSetAxesPos(GAXISSTART,9*rep.width,
         5*rep.height,lims.xmax-lims.xmin-12*rep.width,GXAXIS);
   ggSetAxesScaling(GLINEARTYPE1,5,1.0,6.0,GXAXIS);
   ggSetAxesPos(GAXISSTART,9*rep.width,
         5*rep.height,lims.ymax-lims.ymin-10*rep.height,GYAXIS);
   ggSetAxesScaling(GLINEARTYPE1,8,1.0,9.0,GYAXIS);

/* DRAW GRID */
   ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION);

/* DRAW POLYLINE AND ADD SYMBOLS */
   ggAddGraphPolyline(6,pnts);
   ggAddGraphMarkers(6,pnts,GSTAR,0);

   gSuspendDevice();
   gCloseGino();
   return 0;
}
```

F90 Code

```
!  USE OF COMPLETE GRAPH DRAWING ROUTINE -
! ggPlotGraph()
use gino_f90
use graf_f90

  type (GPOINT), dimension(6) :: pnts = &
    (/GPOINT(1.0,7.0),GPOINT(2.1,8.5),GPOINT(3.1,7.3), &
    GPOINT(4.2,4.1),GPOINT(5.1,5.2),GPOINT(6.0,1.2)/)

  call gOpenGino
  call xxxxx
  call gNewDrawing
  call ggSetGraphCharMode(GGINOMODE)

  call ggPlotGraph(6,pnts,GLINEAR,GLINEAR,-GSTRAIGHT, &
    GFRAME)

  call gSuspendDevice
  call gCloseGino
  stop
  end
```

```
        !  USING LOW LEVEL COMPONENT ROUTINES
        use gino_f90
        use graf_f90

          type (GPOINT), dimension(6) :: pnts = &
            (/GPOINT(1.0,7.0),GPOINT(2.1,8.5),GPOINT(3.1,7.3), &
            GPOINT(4.2,4.1),GPOINT(5.1,5.2),GPOINT(6.0,1.2)/)
          integer flg
          type (GLIMIT) lims
          type (GCHASTY) rep

          call gOpenGino
          call xxxxx
          call ggSetGraphCharMode(GGINOMODE)

        ! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
          call ggEnqPlotFrame(flg,lims)
          call gEnqCharAttribs(rep)

        ! SET UP AXES POSITIONS AND SCALES
          call ggSetAxesPos(GAXISSTART,9.*rep%width, &
            5.*rep%height,lims%xmax-lims%xmin-12.*rep%height,GXAXIS)
          call ggSetAxesScaling(GLINEARTYPE1,5,1.0,6.0,GXAXIS)
          call ggSetAxesPos(GAXISSTART,9.*rep%width, &
            5.*rep%height,lims%ymax-lims%ymin-10.*rep%height,GYAXIS)
          call ggSetAxesScaling(GLINEARTYPE1,8,1.,9.,GYAXIS)

        ! DRAW GRID
          call ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION)

        ! DRAW POLYLINE AND ADD SYMBOLS
          call ggAddGraphPolyline(6,pnts)
          call ggAddGraphMarkers(6,pnts,GSTAR,0)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Graph Scaling

Under the default conditions, the ggPlotGraph() routine will draw a graph representing the full range of the X and Y data, calculating sensible axes intervals according to the current drawing area. The routine ggSetGraphScaling() can be used to add control over the ranges and intervals of the X and Y axes for this type of graph.

ggSetGraphScaling(mode)

where **mode** can be one of the following:

`GDEFAULT`	set the default conditions
`GEQUALLIMITS`	set both axes to have the same limits (to include both data ranges)

`GEQUALRANGES`	set both axes to have the same range of data
`GEQUALGRAPHINTERVALS`	set both axes to have equal sized intervals (in graph coordinates)
`GEQUALSPACEINTERVALS`	set both axes to have equal sized intervals (in space/physical coordinates)

The GEQUALSPACEINTERVALS mode will result in a square grid, but may have the effect of reducing the length of one axis within the specified drawing area in order to maintain sensible numerical intervals on the axes.

Graph Drawing Components

The following describes the low level routines provided by GINOGRAF for drawing graphs. The routines provide the user with the tools to build up more complex and sophisticated graphs than those produced by the Complete Graph Drawing Routine described previously.

Each graph is built up in a modular fashion, making use of individual routines to:-

- Position, scale, label and draw the axes
- Represent data sets with symbols, lines, curves, filled areas and error bars on graphs
- Add further annotation and titles

Graph Axes Definition and Display

Axes definition is essential to graph drawing when creating a graph using the component routines shown later. The axes may be defined using the Complete Drawing routine described previously or through the individual axes definition routines ggSetAxesPos() and ggSetAxesScaling(). A usual requirement for using the individual routines is where the automatic calculation of position or ranges is not sufficient for a complex graph display.

Full description of axes definition, display and titling is found in the Axes section of this manual.

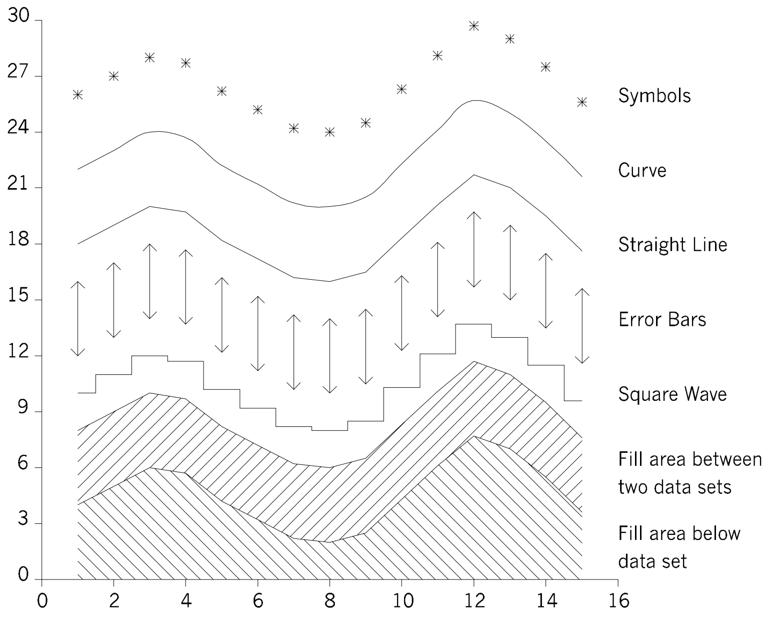
Data Representation

GINOGRAF offers seven forms of data representation on the currently defined graphical axes coordinate system as set up by ggSetAxesPos() and ggSetAxesScaling():

- Straight lines between points
- Smooth curves through points
- Symbols at points
- Error bars at points
- Square wave about points
- Filled areas below sets of points
- Filled areas between two sets of points

If axes have not been defined, data will be plotted according to the default values for axis position and scaling. This could result in points being outside the available drawing area or current window.

All seven output forms obtained from these routines are shown below. Combinations of these routines may be used to superimpose features on top of other graphs, eg, symbols may be drawn at points which are connected by straight lines.



Various Data Representations

Line Graphs

Straight Lines Graphs

ggAddGraphPolyline() is used to draw straight lines between data points.

```
ggAddGraphPolyline(npts,points)
```

where **points** is an array of type GPOINT containing the X and Y parts of the graphical axes coordinates defining the points, while **npts** is the number of points to be joined together.

The routine ggAddGraphPolyline() only draws straight lines between the points, the points themselves are not marked.

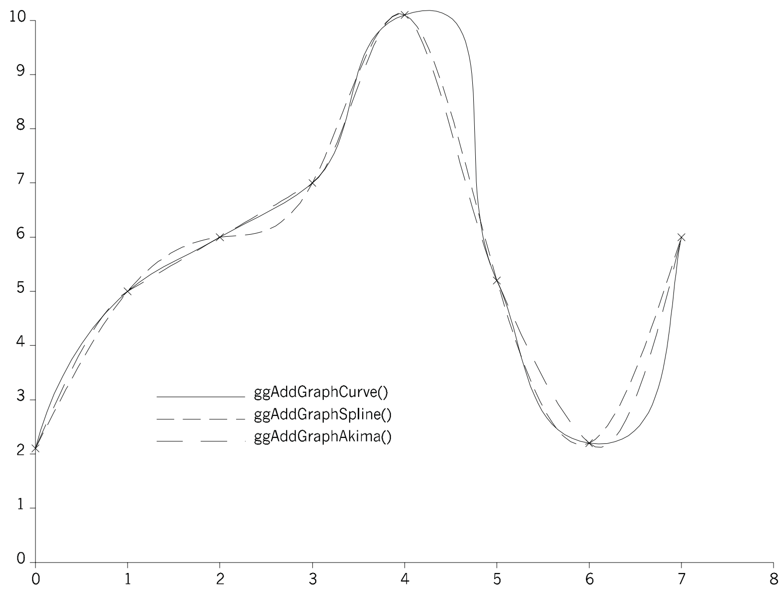
Smooth Curves

There are three forms of smooth curve that can be produced; piecewise cubics, Akima piecewise cubics and piecewise cubic splines. The three routines are:

```
ggAddGraphCurve(npts,points)

ggAddAkimaCurve(npts, points)

ggAddGraphSpline(npts,points)
```

All routines will draw a smooth curve through the points defined in the array **points**, in the order in which they are given (ie, the curve may go back on itself). The three routines use different algorithms for calculating the curve and any may be appropriate depending on the point distribution. In general, ggAddGraphCurve() produces the loosest curve, ggAddAkimaCurve() gives a tighter curve but is less accurate to the fitting function and ggAddGraphSpline() also gives a tighter curve and in most cases is the more acceptable. The tension of the spline curve can also be controlled through the GINO-F routine gSetSplineTension().

A comparison of the different curve drawing routines is illustrated below



Curve Drawing

Curve End Conditions

The two routines to determine the end conditions for all the curve drawing routines are:

ggSetCurveStartConds(beg,cosbeg,sinbeg,xbeg,ybeg)

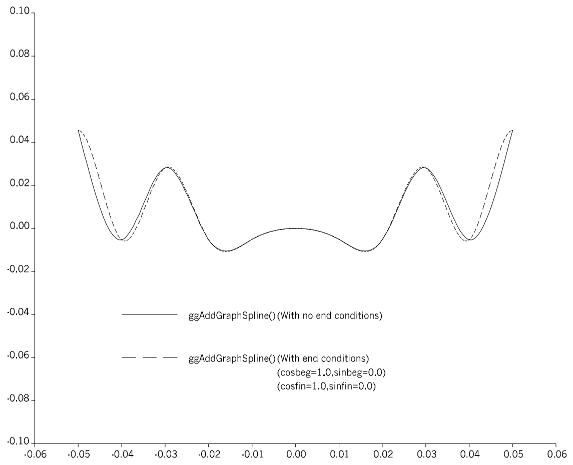
ggSetCurveEndConds(fin,cosfin,sinfin,xfin,yfin)

Each routine is used to set the slope at the beginning or the finish of the curve using either the COSINE and SINE of the required angle or by specifying an extra point in graph coordinates through which the curve would pass if continued.

beg and **fin** are integers determining which method is used. If either argument is equal to GXPOINT, the extra point is used, if either is equal to GNONE, no end conditions are set and if either is equal to GANGLE, the COSINE and SINE arguments are used. **cosbeg**, **sinbeg** and **cosfin**, **sinfin** are the cosine and sine of the required angle if **beg** or **fin** = GANGLE. **xbeg**, **ybeg** and **xfin**, **yfin** specify the extra point if **beg** or **fin** = GXPOINT.

The specified end conditions remain in effect for all curve drawing routines, ggPlotGraph(), ggPlotXYPolarChart(), ggAddGraphCurve(), ggAddAkimaCurve() and ggAddGraphSpline() until reset with **beg** or **fin** being set to GNONE.

The effect of using curve end conditions is shown below.



Use of curve end conditions

Symbol Graphs

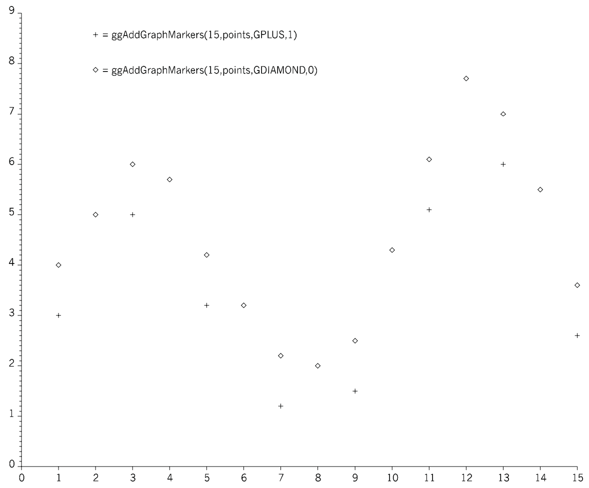
The routine to plot a data set as symbols on a graph (or as a scatter graph) is:

ggAddGraphMarkers(npts,points,nsym,nspace)

ggAddGraphMarkers() considers the **npts** points defined in the array **points**, and then draws the symbol **nsym** at the position of the first point. A symbol is drawn at the first point and then **nspace** points are skipped and the next symbol is drawn. Each successive symbol is then drawn after a further interval of **nspace** points. For example: if **nspace** = 1, symbols will be drawn at points 1,3,5,7 etc. The points are plotted in the order that they exist within the array passed to the routine.

Any of the symbols available through the GINO routine gDrawMarker() are also available to ggAddGraphMarkers(). The standard GINO symbols (0-8), illustrated in Appendix A of this manual, are available to all GINOGRAF users and a large number of additional symbols are also available using the symbol fonts in GINO-F. The size of the symbols is subject to the current character size.

The type of output obtained from ggAddGraphMarkers() is illustrated below. The upper row of symbols are plotted at each point, whereas the second row has **nspace** set to one and so skips every other symbol.



Example output from ggAddGraphMarkers

Error Bars

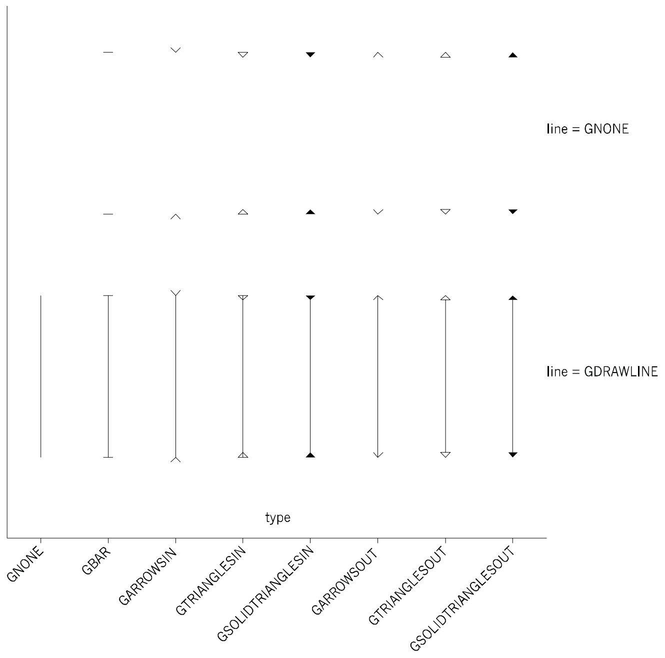
Rather than using a precise point to display a data position an error range (or High-Low range) may be shown. This may be represented by either two symbols, two symbols joined by a line, or just a line. This involves supplying the data point itself and the distance from the data point to the extremes of the range.

The routine to plot Error Bars is:

```
ggAddErrorBars(npts,points,errors,type,line,xory)
```

where the arrays **points** (of type GPOINT) and **errors** (of type GERROR) are of length **npts**. The routine ggAddErrorBars() will plot error bars at each of the points stored in the array **points**, where the range of the errors for each point is stored in the array **errors** (containing both the upper and lower deviation) as relative values (both positive) to the data value. The error values are measured against the axis specified by the argument **xory**, where **xory**=GXAXIS for errors against the Y axis (perpendicular to X axis), and **xory**=GYAXIS for errors against the X axis (perpendicular to Y axis).

The style of the error bars is set using **type** and **line** with the range of possible combinations shown below. The size of the symbols is subject to the current character width.



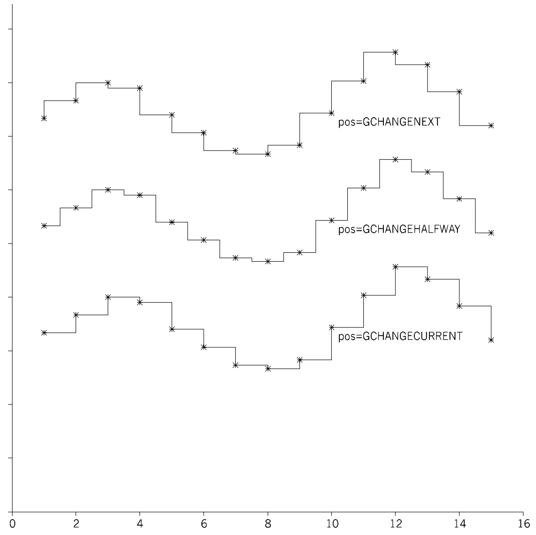
Examples of Error Bar representations

Square Wave Graphs

The routine ggAddSquareWave() draws a square wave about a set of **npts** points stored in the array **points**.

ggAddSquareWave(npts,points,pos,xory)

where **xory** determines which axis the steps are perpendicular to (**xory**=GXAXIS for perpendicular to X axis and **xory**=GYAXIS for perpendicular to Y axis). Control over where the square wave interpolation occurs is determined by **pos** with the 3 possible positions shown below (the actual X and Y points are highlighted by a symbol).



Example of Square Wave Graph

Population Graphs

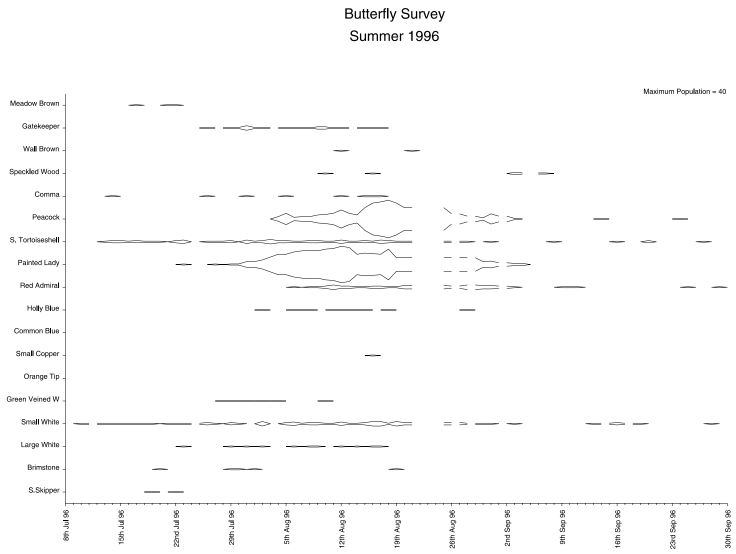
The routine ggAddPopulationGraph() draws a population graph with data at intervals of **dx** at a position **y** on a discrete axis. A set of **npts** recordings are held in the array **points**.

```
ggAddPopulationGraph(dx,y,npts,points,popmax)
```

The population data set is assumed to consist of a series of recordings **points.x**, **points.y** for a species **y**, where **points.x** represents a sample recording point along the non-discrete axis with a standard interval of **dx** and **y** represents a point on the discrete axis. Missing data is represented where the values in the **points.x** array do not follow the specified interval **dx**.

Population values of zero are represented by a point **points.x**,**y**, whereas positive populations are represented by a pair of lines either side of the point **points.x**,**y** joined to the previous recorded data. The value **popmax** controls the scaling of the population graph by setting the maximum population that is to be displayed between the intervals on the discrete axis.

The non-discrete axis can be specified as having date scaling as shown in the example shown below. This graph consists of a series of population graphs with data recorded at irregular daily intervals. The value of **dx** is therefore set at 1.0, with **popmax** set at 40.0.



Graph showing multiple population graphs

Data Set Filling

Two graph filling routines are provided. One fills the area between a data set and a line perpendicular to a point on either the X or Y axis within the defined axes; the second fills the area between two data sets.

The fill style is determined by the combination of **fill** and **line**. Various hatches and cross hatches as well as solid fill are available. If **fill** is negative the area is not filled. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. Further information on line style definition (which includes colour definition) and information on hatch and fill style definition appears in the introduction.

Filling to a Line

The routine ggFillBelowDataset() fills the area between a data set and a line perpendicular to a point on either the X or Y axis.

```
ggFillBelowDataset(npts,points,xylev,xory,fill,line)
```

The area filled is defined by the set of data points stored in the array **points**, the line at **xylev**, and two lines from the first and last data points perpendicular to the line at **xylev**. If **xylev** lies outside one of the graph boundaries then that boundary is used as the edge to which the area is filled.

The area to be filled is made up of the coordinates in the data set in ascending order followed by two points at **xylev** or graph limit. If the boundary crosses over itself in any form, strict polygon filling rules apply and areas outside the two data sets may get filled.

ggFillBelowDataset() may be used on its own to display single data sets, but is also useful for displaying data sets in conjunction with the routine ggFillBetweenDatasets() described below.

Filling Between Two Data Sets

The routine to fill between two data sets held in **xy1** and **xy2** is ggFillBetweenDatasets():

```
ggFillBetweenDatasets(n1,xy1,n2,xy2,fill,line)
```

where **n1** and **n2** are the number of data points held in the arrays **xy1** and **xy2** respectively. **n1** and **n2** do not have to be equal.

The area to be filled is made up of the coordinates in the first data set in ascending order followed by the coordinates in the second data set in descending order. If the boundary crosses over itself in any form, strict polygon filling rules apply and areas outside the two data sets may get filled.

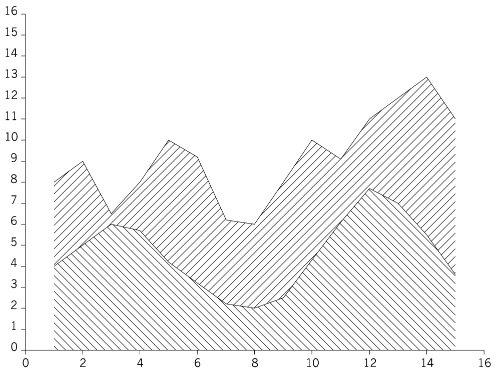
An example of both ggFillBelowDataset() and ggFillBetweenDatasets() is shown below. In this example ggFillBelowDataset() is used with the bounding line set below the X axis, ggFillBetweenDatasets() is then called using a second set of data points and the ones previously used by ggFillBelowDataset(). To emphasize the polygons they are bounded using two calls to ggAddGraphPolyline().

C Code

```
/*  EXAMPLE OF DATA SET FILLING */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GPOINT pnts1[15] = {1.0,4.0,2.0,5.0,3.0,6.0,4.0,5.7,
    5.0,4.2,6.0,3.2,7.0,2.2,8.0,2.0,9.0,2.5,10.0,4.3,
    11.0,6.1,12.0,7.7,13.0,7.0,14.0,5.5,15.0,3.6};

  GPOINT pnts2[15] = {1.0,8.0,2.0,9.0,3.0,6.5,4.0,8.0,
    5.0,10.0,6.0,9.2,7.0,6.2,8.0,6.0,9.0,8.0,10.0,10.0,
    11.0,9.1,12.0,11.0,13.0,12.0,14.0,13.0,15.0,11.0};

  gOpenGino();
  xxxxx();
  gNewDrawing();
  ggSetGraphCharMode(GGINOMODE);

/* DEFINE AXES SCALING AND DRAW AXES */
  ggSetAxesScaling(GLINEARTYPE3,8,0.0,16.0,GXAXIS);
  ggSetAxesScaling(GLINEARTYPE3,16,0.0,16.0,GYAXIS);
  ggDrawAxes(GCARDINAL,GCLOCKWISE,GLOCKWISE,GXAXIS);
  ggDrawAxes(GCARDINAL,GANTICLOCKWISE,GANTICLOCKWISE,GYAXIS);

/* FILL TO BASE LINE */
  ggFillBelowDataset(15,pnts1,0.0,GYAXIS,GFINELEFTDIAGONAL,
        GCURRENT);
  ggAddGraphPolyline(15,pnts1);

/* FILL BETWEEN DATA SETS */
  ggFillBetweenDatasets(15,pnts1,15,pnts2,GFINERIGHTDIAGONAL,
        GCURRENT);
  ggAddGraphPolyline(15,pnts2);

  gSuspendDevice();
  gCloseDevice();
  return(0);
}
```

F90 Code

```
    !  EXAMPLE OF DATA SET FILLING
    use gino_f90
    use graf_f90

      type (GPOINT), dimension(15) :: pnts1 = &
        (/GPOINT(1.0,4.0),GPOINT(2.0,5.0),GPOINT(3.0,6.0), &
          GPOINT(4.0,5.7),GPOINT(5.0,4.2),GPOINT(6.0,3.2), &
          GPOINT(7.0,2.2),GPOINT(8.0,2.0),GPOINT(9.0,2.5), &
          GPOINT(10.0,4.3),GPOINT( 11.0,6.1),GPOINT(12.0,7.7), &
          GPOINT(13.0,7.0),GPOINT(14.0,5.5),GPOINT(15.0,3.6}/)
      type (GPOINT), dimension(15) :: pnts2 = &
        (/GPOINT(1.0,8.0),GPOINT(2.0,9.0),GPOINT(3.0,6.5), &
          GPOINT(4.0,8.0),GPOINT(5.0,10.0),GPOINT(6.0,9.2), &
          GPOINT(7.0,6.2),GPOINT(8.0,6.0),GPOINT(9.0,8.0), &
          GPOINT(10.0,10.0),GPOINT(11.0,9.1),GPOINT(12.0,11.0), &
          GPOINT(13.0,12.0),GPOINT(14.0,13.0),GPOINT(15.0,11.0}/)

      call gOpenGino
      call xxxxx
      call gNewDrawing
      call ggSetGraphCharMode(GGINOMODE)

    ! DEFINE AXES SCALING AND DRAW AXES
      call ggSetAxesScaling(GLINEARTYPE3,8,0.0,16.0,GXAXIS)
      call ggSetAxesScaling(GLINEARTYPE3,16,0.0,16.0,GYAXIS)
      call ggDrawAxes(GCARDINAL,GCLOCKWISE,GLOCKWISE,GXAXIS)
      call ggDrawAxes(GCARDINAL,GANTICLOCKWISE,GANTICLOCKWISE, &
        GYAXIS)

    ! FILL TO BASE LINE
      call ggFillBelowDataset(15,pnts1,0.0,GYAXIS,GFINELEFTDIAGONAL, &
                 GCURRENT)
      call ggAddGraphPolyline(15,pnts1)

    ! FILL BETWEEN DATA SETS
      call ggFillBetweenDatasets(15,pnts1,15,pnts2,GFINERIGHTDIAGONAL, &
                 GCURRENT)
      call ggAddGraphPolyline(15,pnts2)

      call gSuspendDevice
      call gCloseDevice
      stop
      end
```



Example of data set filling

Displaying Data Values

Data points may be annotated with either their X or Y values using the routine ggAddGraphValues():

```
ggAddGraphValues(npts,points,xory)
```

where **points** is an array of type GPOINT containing **npts** coordinate positions which are to be annotated. The argument **xory** determines which of the two values are to be displayed (**xory**=GXAXIS for the X values and **xory**=GYAXIS for the Y values).

By default the values selected are displayed centrally over the corresponding coordinate position using the numeric format of the appropriate axis. Therefore if X values are to be displayed they are output using the same format as values displayed on the X axis. The format is controlled through the routine ggSetAxesAnnotation().

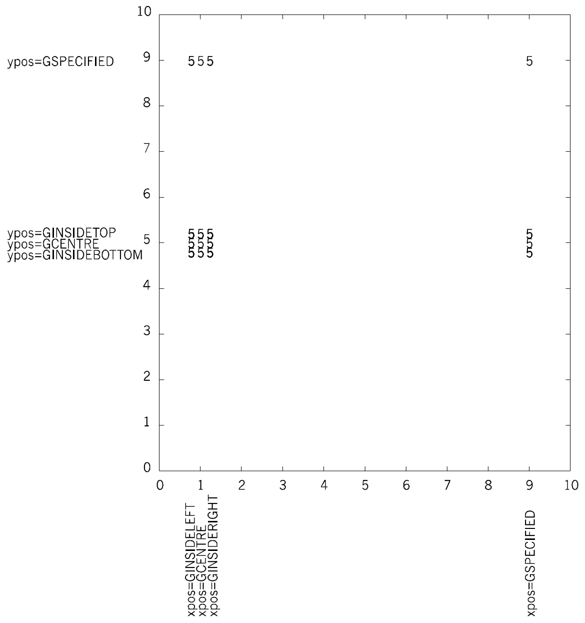
Graph Data Value Control

The default position, orientation, justification of the value displayed can be changed through the routine ggSetValueAttribs():

```
ggSetValueAttribs(xpos,ypos,xory,xoff,yoff,angstr,justmb,juslcr)
```

where **xpos**, **ypos** and **xory** determine the position of each values control point, about which the remaining arguments refer. The control point can be positioned at one of 9 positions around each data point as well as 3 positions at a specified X coordinate (by setting **xpos** to GSPECIFIED), 3 positions at a specified Y coordinate (by setting **ypos** to GSPECIFIED) and at a fixed position by setting both **xpos** and **ypos** to GSPECIFIED. The specified coordinate position of **xory** is measured in graphical coordinates and so refers to the current axes ranges (as set up by ggSetAxesScaling()) for the appropriate axis.

All of the possible control points for a single data value are shown below using a setting of **xory** of 9.0. Each of the data values are drawn centrally over each control point. Note that if both **xpos** and **ypos** are set to GSPECIFIED **all** data values will be output at the same position (X=**xory**,Y=**xory**).



Data value display control positions

The routine ggSetValueAttribs() can also be used to set an additional offset (measured in user space coordinates) in the horizontal and/or vertical direction (**xoff**,**yoff**), an annotation string angle (**angstr**) and a vertical and horizontal justification (**justmb**,**juslcr**). The offsets are measured in user space coordinates and the string angle is measured in degrees (anticlockwise) from the 3 o'clock position.

If the data value is not required to be displayed centrally over the data coordinate position, certain combinations of control point positioning and string justification are required. For example, if data values are required to the right of the data point - setting **xpos** to GOUTSIDERIGHT and setting **juslcr** to left justification will produce the desired effect. Alternatively, if the data values are required to the left of the data point - setting **xpos** to GOUTSIDELEFT and setting **juslcr** to right justification will produce the desired effect. Further examples are shown below.

The current attributes for value charts can be enquired with the routine ggEnqValueAttribs().

ggEnqValueAttribs(xpos,ypos,xory,xoff,yoff,angstr,justmb,juslcr)

Each value in any of the value charts can have a prefix and/or suffix string using the routine ggSetValueTags().

ggSetValueTags(prefix,suffix)

where **prefix** and **suffix** are strings of up to 30 characters that are appended to all the values in one value chart output. The prefix and/or suffix strings are included as part of the value when calculating the justified position of the total output.

An example showing the use of ggSetValueAttribs() and ggSetValueTags() is shown below

C code

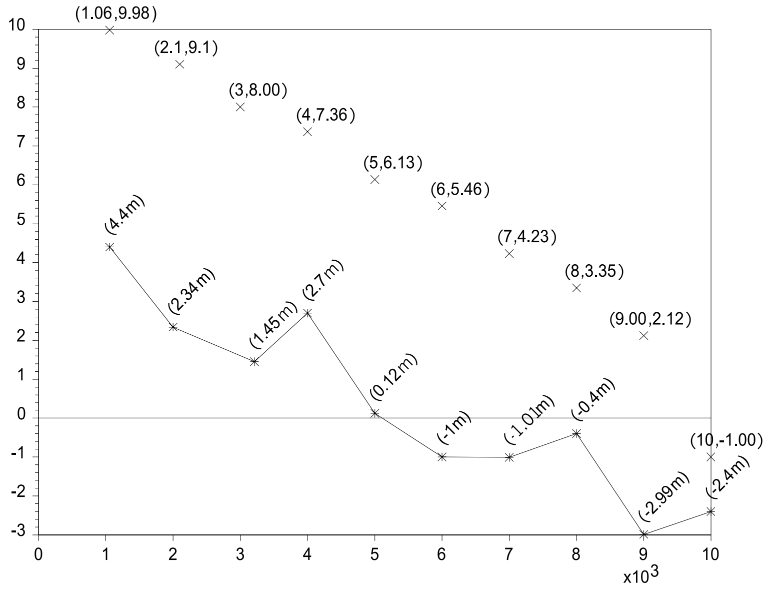
```
/*  DATA VALUE ATTRIBUTE CONTROL */
#include <gino-c.h>
#include <graf-c.h>
#define N 10

int main(void) {
   GDIM paper;
   int papty;
   GPOINT pnts1[N] = {1056.78543,4.4,2000.0,2.34,
    3210.9876,1.454,4000.0,2.7,5000.0,0.123456789012345,
    6000.56,-1.0,7000.0,-1.01,8000.0,-0.4,9000.0,-2.99,
    10000.0,-2.399983456789};

   GPOINT pnts2[N] = {1056.2,9.9823,2100.0,9.0999,
    3000.0,8.003,4000.0,7.365,5000.0,6.134,6000.0,5.4567,
    7000.0,4.23,8000.0,3.345,9000.99,2.123,9999.99,
    -0.9994};

/* SET UP GRID */
   gOpenGino();
   xxxxx();
   gEnqDrawingLimits(&paper,&papty);
   gNewDrawing();

   ggRestoreAxesSettings();
   ggSetGraphCharMode(GGINOMODE);
```

```
/* SET UP GRAPH AXES */
   ggSetAxesPos(GAXISSTART,0.1*paper.xpap,0.1*paper.ypap,
                            0.8*paper.xpap,GXAXIS);
   ggSetAxesScaling(GLINEARTYPE3,10,0.0,9999.99,GXAXIS);
   ggSetAxesPos(GAXISSTART,0.1*paper.xpap,0.1*paper.ypap,
                            0.8*paper.ypap,GYAXIS);
   ggSetAxesScaling(GLINEARTYPE3,13,-3.0,10.0,GYAXIS);
   ggAddGrid(GNONE,GNONE,GNOANNOTATION,GNOANNOTATION);
   ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS);
   ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE,GANTICLOCKWISE,GYAXIS);
/* DRAW FIRST GRAPH ANNOTATING Y VALUES */
   ggAddGraphPolyline(N,pnts1);
   ggAddGraphMarkers(N,pnts1,GSTAR,0);

/* Y VALUES - ABOVE DATA POINT, LEFT JUSTIFIED AT 45 DEGREES */
   ggSetValueAttribs(GCENTRE,GINSIDETOP,0.0,0.0,0.0,45.0,GBOTTOM,
             GLEFT);
   ggSetValueTags("(","m)");
   ggAddGraphValues(N,&pnts1,GYAXIS);

/* DRAW SECOND GRAPH ANNOTATING X AND Y VALUES
   ggAddGraphMarkers(N,&pnts2,GCROSS,0);

/* X VALUES - RIGHT JUSTIFIED WITH '(' PREFIX
   AND ',' SUFFIX OFFSET BY 2MM IN X DIRECTION */
   ggSetValueAttribs(GCENTRE,GINSIDETOP,0.0,2.0,0.0,0.0,GBOTTOM,
              GRIGHT);
   ggSetValueTags("(",",");
   ggAddGraphValues(N,&pnts2,GXAXIS);

/* Y VALUES - LEFT JUSTIFIED WITH ')' SUFFIX
   ALSO OFFSET BY 2MM IN X DIRECTION */
   ggSetValueAttribs(GCENTRE,GINSIDETOP,0.0,2.0,0.0,0.0,GBOTTOM,
                 GLEFT);
   ggSetValueTags("*.",")");
   ggAddGraphValues(N,&pnts2,GYAXIS);

   gSuspendDevice();
   gCloseGino();
```

F90 Code

```
!  DATA VALUE ATTRIBUTE CONTROL
use gino_f90
use graf_f90
  integer, parameter :: N = 10


  type (GDIM) paper
  integer papty
  type (GPOINT), dimension(N) :: pnts1 = &
  (/GPOINT(1056.78543,4.4),GPOINT(2000.0,2.34), &
    GPOINT(3210.9876,1.454),GPOINT(4000.0,2.7), &
    GPOINT(5000.0,0.123456789012345),GPOINT(6000.56,-1.0), &
    GPOINT(7000.0,-1.01),GPOINT(8000.0,-0.4), &
    GPOINT(9000.0,-2.99),(10000.0,-2.399983456789)/)
  type (GPOINT), dimension(N) :: pnts2 = &
  (/GPOINT(1056.2,9.9823),GPOINT(2100.0,9.0999), &
    GPOINT(3000.0,8.003),GPOINT(4000.0,7.365), &
    GPOINT(5000.0,6.134),GPOINT(6000.0,5.4567), &
    GPOINT(7000.0,4.23),GPOINT(8000.0,3.345), &
    GPOINT(9000.99,2.123),GPOINT(9999.99,-0.9994)/)
```

```
         ! SET UP GRID
           call gOpenGino
           call xxxxx
           call gEnqDrawingLimits(paper,papty)
           call gNewDrawing
           call ggRestoreAxesSettings
           call ggSetGraphCharMode(GGINOMODE)

         ! SET UP GRAPH AXES
           call ggSetAxesPos(GAXISSTART,0.1*paper%xpap,0.1*paper%ypap, &
                             0.8*paper%xpap,GXAXIS)
           call ggSetAxesScaling(GLINEARTYPE3,10,0.0,9999.99,GXAXIS)
           call ggSetAxesPos(GAXISSTART,0.1*paper%xpap,0.1*paper%ypap, &
                             0.8*paper%ypap,GYAXIS)
           call ggSetAxesScaling(GLINEARTYPE3,13,-3.0,10.0,GYAXIS)
           call ggAddGrid(GNONE,GNONE,GNOANNOTATION,GNOANNOTATION)
           call ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS)
           call ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE,GANTICLOCKWISE, &
                           GYAXIS)

         ! DRAW FIRST GRAPH ANNOTATING Y VALUES
           call ggAddGraphPolyline(N,pnts1)
           call ggAddGraphMarkers(N,pnts1,GSTAR,0)

         ! Y VALUES - ABOVE DATA POINT, LEFT JUSTIFIED AT 45 DEGREES
           call ggSetValueAttribs(GCENTRE,GINSIDETOP,0.0,0.0,0.0,45.0, &
                          GBOTTOM,GLEFT)
           call ggSetValueTags('(','m)')
           call ggAddGraphValues(N,pnts1,GYAXIS)

         ! DRAW SECOND GRAPH ANNOTATING X AND Y VALUES
           call ggAddGraphMarkers(N,pnts2,GCROSS,0)

         !  X VALUES - RIGHT JUSTIFIED WITH '(' PREFIX AND ',' SUFFIX
         !             OFFSET BY 2MM IN X DIRECTION
           call ggSetValueAttribs(GCENTRE,GINSIDETOP,0.0,2.0,0.0,0.0, &
                          GBOTTOM,GRIGHT)
           call ggSetValueTags('(',',')
           call ggAddGraphValues(N,pnts2,GXAXIS)

         !  Y VALUES - LEFT JUSTIFIED WITH ')' SUFFIX
         !             ALSO OFFSET BY 2MM IN X DIRECTION
           call ggSetValueAttribs(GCENTRE,GINSIDETOP,0.0,2.0,0.0,0.0, &
                          GBOTTOM,GLEFT)
           call ggSetValueTags('*.',')')
           call ggAddGraphValues(N,pnts2,GYAXIS)

           call gSuspendDevice
           call gCloseGino
           stop
           end
```



Displaying data values with format control

Application Specific Missing Values

GINOGRAF provides a facility for catering for a single value or range of values that represent missing or rogue data in an application’s data set. This facility is provided through the routine ggDefineMissingValues().

ggDefineMissingValues(mode, val1,val2,xory)

where **mode** sets the missing value mode to one of the following:

0 - switches the facility off
1 - omits data that is equal to **val1** or **val2**
2 - omits data greater that **val1**
3 - omits data greater than or equal to **val1**
4 - omits data less than **val1**
5 - omits data less than or equal to **val1**
6 - omits data outside the range **val1-val2**
7 - omits data inside the range **val1-val2**

This facility only affects polyline, marker and value graphs either drawn through ggPlotGraph(), ggAddGraphPolyline(), ggAddGraphMarkers() or ggAddGraphValues().

Graph Titles

The titling routine ggDrawGraphTitle() may be used to title a graph. Further details of this routine and other graphing utilities can be found in the Utilities chapter.

Chapter 4

CHARTS

Introduction to Charts

This chapter covers the production of Charts. The GINOGRAF definition of a chart is a two or two-and-a-half dimensional (block-filled) representation of data using rectangular columns, bars, steps or areas on a set of axes. This definition covers five main types of chart: Multi-column and Single column Histograms and Bar Charts (discrete values against continuous data); and Step Charts and Area Charts (continuous data and values).

Data for the five chart types is represented by lengths, heights or areas as shown below :

	Column Width	**First data position**	**Second data position**	**Data represented by**
Histograms	Constant	Zero	Variable	Height
Bar Charts	Constant	Variable	Variable	Length
Step Charts	Variable	Fixed	Variable	Height and width
Area Charts	Variable	Variable	Variable	Length and width
Multi-data Set Histograms	Constant	Zero	Variable	Height

The chart drawing routines are split into two main groups: complete drawing routines which produce a fully annotated chart with a single routine call; and separate component routines which build up a chart in a modular fashion. The routine names are shown in the tables below:

	Complete
Histogram	ggPlotHistogram
Bar Charts	ggPlotBarChart
Step Charts	ggPlotStepChart
Area Charts	ggPlotAreaChart

	Outline	Value Display
Histogram	ggAddHistogramOutline	ggAddHistogramValues
Bar Charts	ggAddBarChartOutline	ggAddBarChartValues
Step Charts	ggAddStepChartOutline	ggAddStepChartValues
Area Charts	ggAddAreaChartOutline	ggAddAreaChartValues

	Filling	Block Filling
Histogram	ggFillHistogram	ggBlockFillHistogram
Bar Charts	ggFillBarChart	ggBlockFillBarChart
Step Charts	ggFillStepChart	ggBlockFillStepChart
Area Charts	ggFillAreaChart	ggBlockFillAreaChart
Multi-Histogram	ggFillMultiHistogram	ggBlockFillMultiHistogram

While axis and graph titles can be added after the complete drawing routines have been called, the full flexibility of layout and style can only be achieved using the component routines as the complete drawing routine is provided to present user data as quickly as possible with the minimum of effort.

Complete Chart Drawing

Complete chart drawing routines provide the user with the simplest path to obtaining output from GINOGRAF.

The routines described in this section are complete in themselves. The user simply provides the data in the appropriate array and makes a single routine call. The position and scaling of the chart is calculated automatically and output is drawn to fit the current graph drawing area as described in the main introduction. The numerical annotation on each axes is subject to the current settings of format and annotation control set by ggSetAxesAnnotation() and ggSetAxesAttribs().

Histograms

The complete drawing routine to produce a 2D Histogram is:

ggPlotHistogram(ncols,yarray,frac,scy,vbeg,vend)

This routine displays a data set made up of **ncols** values in the array **yarray** as **ncols** rectangular columns drawn between zero and the value in **yarray**.

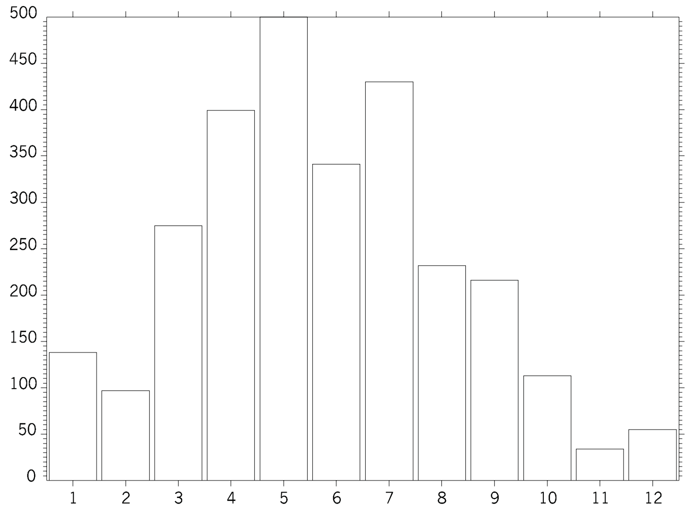
The discrete axis is always drawn as the X axis having a range from **vbeg** to **vend** divided into **ncols** intervals. The Y axis covers the extent of all the values in the array **yarray** and may be drawn with linear or logarithmic scaling as defined by the parameter **scy**(GLINEAR=linear, GLOG10=logarithmic).

The routine allows control over the width of the Histogram bars by specifying the fraction **frac** of the largest possible bar width for the axis definition.

width of columns = ((length of discrete (X) axis)/**ncols**) * **frac**

The effect of various values of **frac** is shown under Histogram outline.

An example of output produced by ggPlotHistogram() is shown below:



Example of complete 2D Histogram

The following code shows the program that generated the above example followed by an equivalent program using the low-level component routines described in the following sections. The comparison is given because it is often desirable to make changes to the layout of a basic Histogram, but this is only possible by using the component routines, as the routine ggPlotHistogram() is provided to present user data as quickly as possible with the minimum of effort.

C Code

```
/* USE OF COMPLETE HISTOGRAM DRAWING ROUTINE -
   ggPlotHistogram() */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   float y[12] = {138.0,97.0,275.0,399.0,500.0,
      341.0,430.0,232.0,216.0,113.0,34.0,55.0};

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

   ggPlotHistogram(12,y,0.9,GLINEAR,1.0,12.0);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

```
/* USING LOW LEVEL COMPONENT ROUTINES*/
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   float y[12] = {138.0,97.0,275.0,399.0,500.0,
       341.0,430.0,232.0,216.0,113.0,34.0,55.0};
    GCHASTY rep;
   GLIMIT lims;

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
   ggEnqPlotFrame(&flg,&lims);
   gEnqCharAttribs(&rep);

/* SET UP AXES POSITIONS AND SCALES */
   ggSetAxesPos(GAXISSTART,9.0*rep.cw,5.0*rep.ch,
          lims.xmax-lims.xmin-12.0*rep.cw,GXAXIS);
   ggSetAxesScaling(GDISCRETE,12,1.0,12.0,GXAXIS);
   ggSetAxesPos(GAXISSTART,9.0*rep.cw,5.0*rep.ch,
          lims.ymax-lims.ymin-10.*rep.ch,GYAXIS);
   ggSetAxesScaling(GLINEARTYPE1,10,0.0,500.0,GYAXIS);

/* DRAW GRID */
   ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION);

/* DRAW HISTOGRAM */
   ggAddHistogramOutline(12,y,0.9);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 Code

```
! USE OF COMPLETE HISTOGRAM DRAWING ROUTINE -
! ggPlotHistogram()
use gino_f90
use graf_f90

  real, dimension(12) :: y = (/138.0,97.0,275.0,399.0, &
    500.0,341.0,430.0,232.0,216.0,113.0,34.0,55.0/)

  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

  call ggPlotHistogram(12,y,0.9,GLINEAR,1.0,12.0)

  call gSuspendDevice
  call gCloseGino
  stop
  end
```

```
! USING LOW LEVEL COMPONENT ROUTINES
use gino_f90
use graf_f90


  real, dimension(12) :: y = (/138.0,97.0,275.0,399.0, &
    500.0,341.0,430.0,232.0,216.0,113.0,34.0,55.0/)
  type (GCHASTY) rep
  type (GLIMIT) lims

  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
  call ggEnqPlotFrame(flg,lims)
  call gEnqCharAttribs(rep)

! SET UP AXES POSITIONS AND SCALES
  call ggSetAxesPos(GAXISSTART,9.0*rep%cw, &
          5.0*rep%ch,lims%xmax-lims%xmin-12.0*rep%cw,GXAXIS)
  call ggSetAxesScaling(GDISCRETE,12,1.0,12.0,GXAXIS)
  call ggSetAxesPos(GAXISSTART,9.0*rep%cw, &
          5.0*rep%ch,lims%ymax-lims%ymin-10.*rep%ch,GYAXIS)
  call ggSetAxesScaling(GLINEARTYPE1,10,0.0,500.0,GYAXIS)

! DRAW GRID
  call ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION)

! DRAW HISTOGRAM
  ggAddHistogramOutline(12,y,0.9)

  gSuspendDevice
  gCloseGino
  stop
  end
```

Bar Charts

The complete drawing routine to produce a 2D Bar Chart is:

ggPlotBarChart(nbars,bars,frac,scx,scy,vbeg,vend)

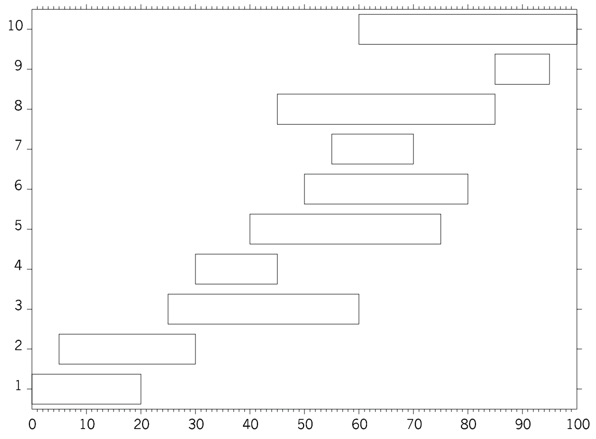
where **nbars** is the number of data pairs contained in the array **bars** (of type GBARCHART) which contain the start and finish values of each bar. The type of scaling and the orientation of the continuous axis (axis defining bar length) is defined by the parameters **scx** and **scy**. The discrete axis is annotated with **nbars** intervals between **vbeg** and **vend**.

The width of the bars is determined by setting **frac**, the fraction of the widest possible bar given the length of axis and the number of bars.

width of bars = ((length of discrete axis)/**nbars**) * **frac**

If **frac** = 1.0 then only the necessary lines are drawn (ie, lines common to two bars are omitted).

An example of output produced by ggPlotBarChart() is shown below:



Example of complete 2D Bar Chart

The following code shows the program that generated the above example followed by an equivalent program using the low-level component routines described in the following sections. The comparison is given because it is often desirable to make changes to the layout of a basic Bar Chart, but this is only possible by using the low level routines, as the routine ggPlotBarChart() is provided to present user data as quickly as possible with the minimum of effort.

C Code

```
/* USE OF COMPLETE BARCHART DRAWING ROUTINE -
   ggPlotBarChart() */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   GBARCHART bars[10] = {0.0,20.0,5.0,30.0,25.0,60.0,
       30.0,45.0,40.0,75.0,50.0,80.0,55.0,70.0,45.0,85.0,
       85.0,95.0,60.0,100.0};

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

   ggPlotBarChart(10,bars,0.75,GLINEAR,GDISCRETE,1.0,10.0);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

```
/* USING LOW LEVEL COMPONENT ROUTINES */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   int flg;
   GLIMIT lims;
   GCHASTY rep;
   GBARCHART bars[10] = {0.0,20.0,5.0,30.0,25.0,60.0,
       30.0,45.0,40.0,75.0,50.0,80.0,55.0,70.0,45.0,85.0,
       85.0,95.0,60.0,100.0};

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
   ggEnqPlotFrame(&flg,&lims);
   gEnqCharAttribs(&rep);

/* SET UP AXES POSITIONS AND SCALES */
   ggSetAxesPos(GAXISSTART,9.0*rep.cw,
          5.0*rep.ch,lims.xmax-lims.xmin-12.0*rep.cw,GXAXIS);
   ggSetAxesScaling(GLINEARTYPE1,11,0.0,100.0,GXAXIS);
   ggSetAxesPos(GAXISSTART,9.0*rep.cw,
          5.0*rep.ch,lims.ymax-lims.ymin-10.0*rep.ch,GYAXIS);
   ggSetAxesScaling(GDISCRETE,10,1.0,10.0,GYAXIS)

/* DRAW GRID */
   ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION);

/* DRAW BAR CHART */
   ggAddBarChartOutline(10,bars,0.75);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 code

```
        !  USE OF COMPLETE BARCHART DRAWING ROUTINE -
        ! ggPlotBarChart()
        use gino_f90
        use graf_f90

          type (GBARCHART), dimension(10) :: bars = &
            (/GBARCHART(0.0,20.0),GBARCHART(5.0,30.0), &
              GBARCHART(25.0,60.0),GBARCHART(30.0,45.0), &
              GBARCHART(40.0,75.0),GBARCHART(50.0,80.0), &
              GBARCHART(55.0,70.0),GBARCHART(45.0,85.0), &
              GBARCHART(85.0,95.0),GBARCHART(60.0,100.0)/)

          call gOpenGino
          call xxxxx
          call ggSetGraphCharMode(GGINOMODE)

          call ggPlotBarChart(10,bars,0.75,GLINEAR,GDISCRETE,1.0,10.0)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

```
        ! USING LOW LEVEL COMPONENT ROUTINES
        use gino_f90
        use graf_f90

          integer flg
          type (GLIMIT) lims
          type (GCHASTY) rep
          type (GBARCHART), dimension(10) :: bars = &
            (/GBARCHART(0.0,20.0),GBARCHART(5.0,30.0), &
              GBARCHART(25.0,60.0),GBARCHART(30.0,45.0), &
              GBARCHART(40.0,75.0),GBARCHART(50.0,80.0), &
              GBARCHART(55.0,70.0),GBARCHART(45.0,85.0), &
              GBARCHART(85.0,95.0),GBARCHART(60.0,100.0)/)

          call gOpenGino
          call xxxxx
          call ggSetGraphCharMode(GGINOMODE)

        ! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
          call ggEnqPlotFrame(flg,lims)
          call gEnqCharAttribs(rep)

        ! SET UP AXES POSITIONS AND SCALES
          call ggSetAxesPos(GAXISSTART,9.0*rep%cw, &
            5.0*rep%ch,lims%xmax-lims%xmin-12.0*rep%cw,GXAXIS)
          call ggSetAxesScaling(GLINEARTYPE1,11,0.0,100.0,GXAXIS)
          call ggSetAxesPos(GAXISSTART,9.0*rep%cw, &
            5.0*rep%ch,lims%ymax-lims%ymin-10.0*rep%ch,GYAXIS)
          call ggSetAxesScaling(GDISCRETE,10,1.0,10.0,GYAXIS)

        ! DRAW GRID
          call ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION)
        ! DRAW BAR CHART
          call ggAddBarChartOutline(10,bars,0.75)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Step Charts & Variable Width Histograms

The Complete Drawing routine to produce a Step Chart is:

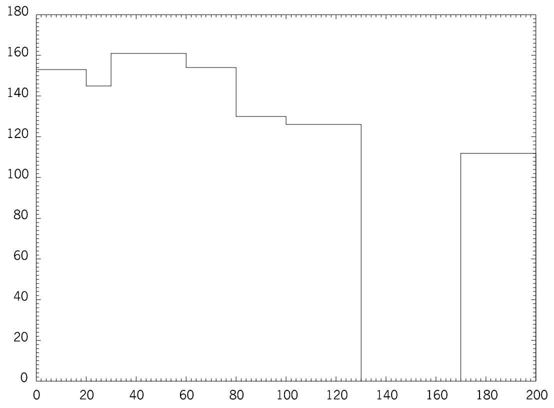
ggPlotStepChart(nsteps,steps,base,scx,scy,drop)

This routine displays an annotated set of axes scaled automatically to include all the data. The data set held in the array **steps** (of type GSTEPCHART) is represented as a series of steps between corresponding values of **steps.s** and **steps.f** on the X axis, at a height held in **steps.h** on the Y axis, or a series of variable width columns between corresponding values of **steps.s** and **steps.f** on the X axis with a height shown between a base value (**base**) and the corresponding value held in **steps.h** on the Y axis.

The choice of step or column is determined by the parameter **drop**, where **drop**= GDROPTYPE0 displays only the step heights, **drop**=GDROPTYPE1 displays the vertical between adjacent steps (ie, **steps.s**(i) = **steps.f**(i)), **drop**=GDROPTYPE2 displays step edges except between adjacent steps (**steps.s**(i) = **steps.f**(i)) and drops down to **base** at the edge of a set of adjacent steps and **drop**=GDROPTYPE3 displays columns down to **base** for every height.

Linear or logarithmic scaling may be used on either axis depending on the value of **scx** and **scy**, with **scx**/**scy**=GLINEAR producing linear scaling on both axes.

An example of output produced by ggPlotStepChart() (with **drop** = GDROPTYPE2) is shown below:



Example of complete 2D Step Chart

The following code shows the program that generated the above example followed by an equivalent program using the low-level component routines described in the following sections. The comparison is given because it is often desirable to make changes to the layout of a basic Step Chart, but this is only possible by using the low level routines, as the routine ggPlotStepChart() is provided to present user data as quickly as possible with the minimum of effort.

C Code

```
/* USE OF COMPLETE STEPCHART DRAWING ROUTINE - ggPlotStepChart() */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   GSTEPCHART steps[7] = {0.0,20.0,153.0,
       20.0,30.0,145.0,30.0,60.0,161.0,
       60.0,80.0,154.0,80.0,100.0,130.0,
       100.0,130.0,126.0,170.0,200.0,112.0};

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

   ggPlotStepChart(7,steps,0.0,GLINEAR,GLINEAR,GDROPTYPE2);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

```
/* USING LOW LEVEL COMPONENT ROUTINES */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   GSTEPCHART steps[7] = {0.0,20.0,153.0,
       20.0,30.0,145.0,30.0,60.0,161.0,
       60.0,80.0,154.0,80.0,100.0,130.0,
       100.0,130.0,126.0,170.0,200.0,112.0};
   GLIMIT lims;
   int flg;

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
   ggEnqPlotFrame(&flg,&lims);
   gEnqCharAttribs(&rep);

/* SET UP AXES POSITIONS AND SCALES */
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.xmax-lims.xmin-12.0*rep.width,GXAXIS);
   ggSetAxesScaling(GLINEARTYPE1,11,0.0,200.0,GXAXIS);
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.ymax-lims.ymin-10.0*rep.height,GYAXIS);

   ggSetAxesScaling(GLINEARTYPE1,10,0.0,180.0,GYAXIS);

/* DRAW GRID */
   ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION);

/* DRAW STEP CHART */
   ggAddStepChartOutline(7,steps,0.0,GDROPTYPE2,GXAXIS);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 Code

```
!  USE OF COMPLETE STEPCHART DRAWING ROUTINE -
! ggPlotStepChart()
use gino_f90
use graf_f90

  type (GSTEPCHART), dimension(7) :: steps = &
    (/GSTEPCHART(0.0,20.0,153.0), &
      GSTEPCHART(20.0,30.0,145.0), &
      GSTEPCHART(30.0,60.0,161.0), &
      GSTEPCHART(60.0,80.0,154.0), &
      GSTEPCHART(80.0,100.0,130.0), &
      GSTEPCHART(100.0,130.0,126.0), &
      GSTEPCHART(170.0,200.0,112.0)/)

  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

  call ggPlotStepChart(7,steps,0.0,GLINEAR,GLINEAR,GDROPTYPE2)
```

```
           call gSuspendDevice
           call gCloseGino
           stop
           end
```

```
         ! USING LOW LEVEL COMPONENT ROUTINES
         use gino_f90
         use graf_f90

            type (GSTEPCHART), dimension(7) :: steps = &
             (/GSTEPCHART(0.0,20.0,153.0), &
               GSTEPCHART(20.0,30.0,145.0), &
               GSTEPCHART(30.0,60.0,161.0), &
               GSTEPCHART(60.0,80.0,154.0), &
               GSTEPCHART(80.0,100.0,130.0), &
               GSTEPCHART(100.0,130.0,126.0), &
               GSTEPCHART(170.0,200.0,112.0)/)
           type (GLIMIT) lims
           integer flg

           call gOpenGino
           call xxxxx
           call ggSetGraphCharMode(GGINOMODE)

         ! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
           call ggEnqPlotFrame(flg,lims)
           call gEnqCharAttribs(rep)

         ! SET UP AXES POSITIONS AND SCALES
           call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
                5.0*rep%height,lims%xmax-lims%xmin-12.0*rep%width,GXAXIS)
           call ggSetAxesScaling(GLINEARTYPE1,11,0.0,200.0,GXAXIS)
           call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
                5.0*rep%height,lims%ymax-lims%ymin-10.0*rep%height,GYAXIS)
           call ggSetAxesScaling(GLINEARTYPE1,10,0.0,180.0,GYAXIS)

         ! DRAW GRID
           call ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION);

         ! DRAW STEP CHART
           call ggAddStepChartOutline(7,steps,0.0,GDROPTYPE2,GXAXIS);

           call gSuspendDevice
           call gCloseGino
           stop
           end
```

Area Charts

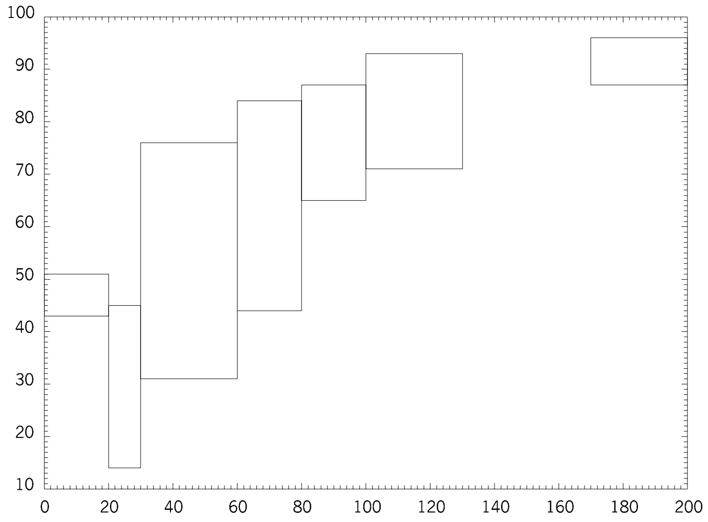
The Complete Drawing routine to produce an Area Chart is:

ggPlotAreaChart(nareas,areas,scx,scy)

This routine displays an annotated set of axes scaled automatically to include all the data. The data set held in the array **areas** (of type GAREACHART) is represented as **nareas** areas between the corresponding coordinates **areas.s**, **areas.h1** and **areas.f**, **areas.h2** where **areas.s** and **areas.f** are positions on the X axis, and **areas.h1** and **areas.h2** are on the Y axis.

Linear or logarithmic scaling may be used on either axis depending on the value of **scx** and **scy**, with **scx**=**scy**=GLINEAR producing linear scaling on both axes.

An example of output produced by ggPlotAreaChart() is shown below:



Example of complete 2D Area Chart

The following code shows the program that generated the above followed by an equivalent program using the low-level component routines described in the following sections. The comparison is given because it is often desirable to make changes to the layout of a basic Area Chart, but this is only possible by using the low level routines, as the routine ggPlotAreaChart() is provided to present user data as quickly as possible with the minimum of effort.

C Code

```
/* USE OF COMPLETE AREA CHART DRAWING ROUTINE
   ggPlotAreaChart() */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   GAREACHART area[7] = {0.0,20.0,43.0,51.0,
       20.0,30.0,14.0,45.0,30.0,60.0,31.0,76.0,
       60.0,80.0,44.0,84.0,80.0,100.0,65.0,87.0,
       100.0,130.0,71.0,93.0,170.0,200.0,87.0,96.0};

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

   ggPlotAreaChart(7,area,GLINEAR,GLINEAR);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

```
/* USING LOW LEVEL COMPONENT ROUTINES */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   GAREACHART area[7] = {0.0,20.0,43.0,51.0,
       20.0,30.0,14.0,45.0,30.0,60.0,31.0,76.0,
       60.0,80.0,44.0,84.0,80.0,100.0,65.0,87.0,
       100.0,130.0,71.0,93.0,170.0,200.0,87.0,96.0};

   int flg;
   GLIMIT lims;
   GCHASTY rep;

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
   ggEnqPlotFrame(&flg,&lims);
   gEnqCharAttribs(&rep);

/* SET UP AXES POSITIONS AND SCALES */
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.xmax-lims.xmin-12.0*rep.width,GXAXIS);
   ggSetAxesScaling(GLINEARTYPE1,11,0.0,200.0,GXAXIS);
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.ymax-lims.ymin-10.0*rep.height,GYAXIS);

   ggSetAxesScaling(GLINEARTYPE1,10,10.0,100.0,GYAXIS);

/* DRAW GRID */
   ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION);

/* DRAW AREA CHART */
   ggAddAreaChartOutline(7,area,GXAXIS);
```

```
            gSuspendDevice();
            gCloseGino();
            return(0);
         }
```

F90 Code

```
         ! USE OF COMPLETE AREA CHART DRAWING ROUTINE
         ! ggPlotAreaChart()
         use gino_f90
         use graf_f90


           type (GAREACHART), dimension(7) :: area = &
             (/GAREACHART(0.0,20.0,43.0,51.0), &
               GAREACHART(20.0,30.0,14.0,45.0), &
               GAREACHART(30.0,60.0,31.0,76.0), &
               GAREACHART(60.0,80.0,44.0,84.0), &
               GAREACHART(80.0,100.0,65.0,87.0), &
               GAREACHART(100.0,130.0,71.0,93.0), &
               GAREACHART(170.0,200.0,87.0,96.0)/)

           call gOpenGino
           call xxxxx
           call ggSetGraphCharMode(GGINOMODE)

           call ggPlotAreaChart(7,area,GLINEAR,GLINEAR)

           call gSuspendDevice
           call gCloseGino
           stop
           end
```

```
         ! USING LOW LEVEL COMPONENT ROUTINES
         use gino_f90
         use graf_f90

           type (GAREACHART), dimension(7) :: area = &
             (/GAREACHART(0.0,20.0,43.0,51.0), &
               GAREACHART(20.0,30.0,14.0,45.0), &
               GAREACHART(30.0,60.0,31.0,76.0), &
               GAREACHART(60.0,80.0,44.0,84.0), &
               GAREACHART(80.0,100.0,65.0,87.0), &
               GAREACHART(100.0,130.0,71.0,93.0), &
               GAREACHART(170.0,200.0,87.0,96.0)/)
           integer flg
           type (GLIMIT) lims
           type (GCHASTY) rep

           call gOpenGino
           call xxxxx
           call ggSetGraphCharMode(GGINOMODE)

         ! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
           call ggEnqPlotFrame(flg,lims)
           call gEnqCharAttribs(rep)
```

```
        ! SET UP AXES POSITIONS AND SCALES
          call ggSetAxesPos(GAXISSTART,9.*rep%width, &
               5.*rep%height,lims%xmax-lims%xmin-12.*rep%width,GXAXIS)
          call ggSetAxesScaling(GLINEARTYPE1,11,0.,200.,GXAXIS)
          call ggSetAxesPos(GAXISSTART,9.*rep%width, &
               5.*rep%height,lims%ymax-lims%ymin-10.*rep%height,GYAXIS)
          call ggSetAxesScaling(GLINEARTYPE1,10,10.,100.,GYAXIS)

        ! DRAW GRID
          call ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION)

        ! DRAW AREA CHART
          call ggAddAreaChartOutline(7,area,GXAXIS)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Chart Scaling

Under the default conditions, the ggPlotStepChart() and ggPlotAreaChart() routines will draw a chart representing the full range of the X and Y data, calculating sensible axes intervals according to the current drawing area. The routine ggSetGraphScaling() can be used to add control over the ranges and intervals of the X and Y axes for this type of chart.

ggSetGraphScaling(mode)

where **mode** can be one of the following:

GDEFAULT	set the default conditions
GEQUALLIMITS	set both axes to have the same limits (to include both data ranges)
GEQUALRANGES	set both axes to have the same range of data
GEQUALGRAPHINTERVALS	set both axes to have equal sized intervals (in graph coordinates)
GEQUALSPACEINTERVALS	set both axes to have equal sized intervals (in space/physical coordinates)

The GEQUALSPACEINTERVALS mode will result in a square grid, but may have the effect of reducing the length of one axis within the specified drawing area in order to maintain sensible numerical intervals on the axes.

Chart Drawing Components

The following sections describes the component routines provided by GINOGRAF for drawing charts. These routines provide the user with the tools to build up more complex and sophisticated charts than those produced by the Complete Chart Drawing routines described in the previous section.

Each chart is built up in a modular fashion, making use of individual routines to:

- Position, scale, label and draw the axes
- Represent data sets in outline or filled areas
- Annotate data sets

Chart Axes Definition and Display

Axes definition is essential to chart drawing when creating a chart using the component routines in the following section. The axes may be defined using the Complete Drawing Routines described previously or through the individual axes definition routines ggSetAxesPos() and ggSetAxesScaling(). A usual requirement for using the individual routines is where the automatic calculation of position or ranges is not sufficient for a complex chart display.

Full description of axes definition, display and titling is found elsewhere (see page 19).

Block Chart Attributes

The format of the two-and-a-half dimensional block charts is subject to the current Block Chart attributes. These are set and enquired using the following two routines:

ggSetBlockChartAttribs(coloff,azim,elev,depth,top,side)

ggEnqBlockChartAttribs(<u>coloff</u>,<u>azim</u>,<u>elev</u>,<u>depth</u>,<u>top</u>,<u>side</u>)

where **coloff** is the colour index offset used to define the block shading colours. The argument **azim** and **elev** define the azimuth and elevation of the block and **depth** is the depth as a fraction of the block width. The variables **top** and **side** define the relative lightness of the top and side of each block. The colour of the front of the block is defined in the block filling routine itself for each chart type.

The default Block Chart attributes may be restored using the routine:

ggRestoreBlockChartAttribs

Histogram Components

There are four components available for the building of a fully annotated, filled Histogram:

`ggBlockFillHistogram()`	Drawing block filled columns
`ggAddHistogramOutline()`	Drawing column outline
`ggFillHistogram()`	Drawing filled columns
`ggAddHistogramValues()`	Displaying height values

All four routines may be used independently of each other or on the same chart although it is usual to use a combination of block filling and values, or simple filling, outline and values. It is necessary to define both X and Y axis positions and data ranges before calling any of these routines either by the axis definition routines or the Complete Chart Drawing routine ggPlotHistogram(). One of the axes should be defined as a discrete axis (**scale**=GDISCRETE) whereupon the heights are measured against the remaining non-discrete axis.

Users should note that if ggFillHistogram() is called after ggPlotHistogram() or ggAddHistogramOutline() the Histogram outline will be overwritten by the filling. If the outline is required, the solution is to always draw the outline with ggAddHistogramOutline() after the filling. The block filling routine fills the appropriate areas and follows this with drawing the histogram outline in the current GINO line colour.

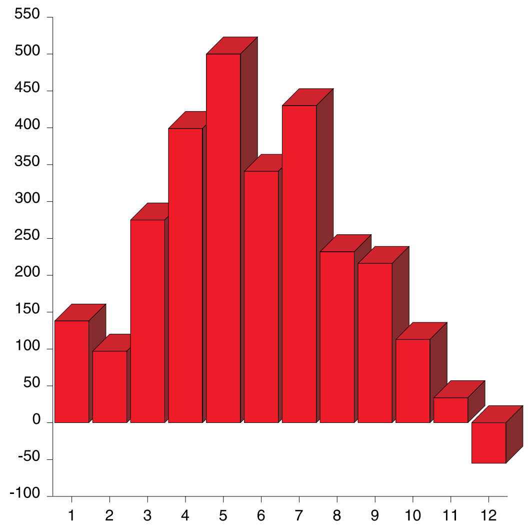
Block Filled Histogram

The routine to display a block filled histogram is:

ggBlockFillHistogram(ncols,yarray,frac,line)

This routine displays a set of data values as block filled columns on the last defined set of axes. The columns are plotted about the centre of the tick marks along the discrete axis according to the current block chart attributes. All the columns are solid filled with the specified **line** style with the extrusions filled with a darker (or lighter) shade of this line style again according to the current block chart attributes (using ggSetBlockChartAttribs()).

The following shows an example of a block filled histogram.



Block Filled Histogram

Histogram Outline

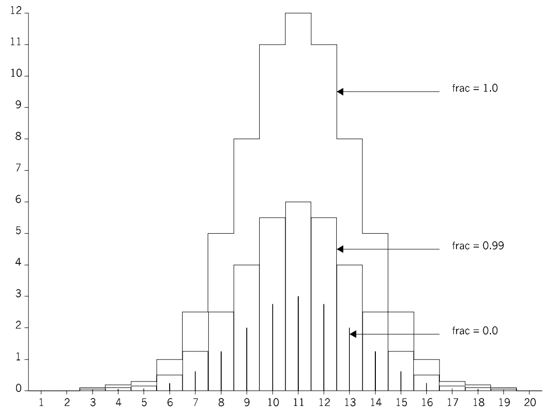
The routine to display a set of columns is:

ggAddHistogramOutline(ncols,yarray,frac)

This routine displays a set of data values as columns on the last defined set of axes. The columns are plotted about the centre of the tick marks along the discrete axis in the order that they are presented within the array.

The representation of the columns depends on the width of the columns which is determined using the parameter **frac**, the fraction of the widest possible column given the axis length and number of columns.

The effects of changing **frac** within ggAddHistogramOutline() are shown below:



Effect of frac on ggAddHistogramOutline

If a height defined in array **yarray** is negative, the column is drawn on the negative side of the discrete axis.

Histogram Filling

The routine to display a set of filled columns is:

ggFillHistogram(ncols,yarray,frac,fill,line)

This routine displays a set of data values as filled columns on the last defined set of axes. The columns are plotted about the centre of the tick marks along the discrete axis in the order that they are presented within the array.

The width of the columns is determined using the parameter **frac**, the fraction of the widest possible column given the axis length and number of columns.

The hatch or fill style for each column is held in the array **fill**. Various hatches and cross hatches as well as solid fill are available. If an element of **fill** contains a number less than -1(GHOLLOW), the corresponding column is not filled and the boundary not drawn. The line styles used to fill the columns are held in the array **line**. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. Further information on line style definition (which includes colour definition) and information on hatch and fill style definition appears in the main introduction.

If a height defined in array **yarray** is negative, the column is filled on the negative side of the discrete axis.

The column outlines are not drawn with this routine. Users should use ggPlotHistogram() or ggAddHistogramOutline() to draw the outline.

Annotating Height Values on Histograms

The routine to output Histogram height values is:

```
ggAddHistogramValues(ncols,yarray,frac)
```

where ggAddHistogramValues() will display **ncols** values in the array **yarray** associated with the Histogram columns drawn by ggPlotHistogram() or ggAddHistogramOutline(). The argument **frac** is required to ensure correct positioning of the values when they are placed at the edge of each column as shown below, it should therefore be set to the same value as that used in a corresponding call to ggPlotHistogram() or ggAddHistogramOutline(). By default, the values are positioned at the centre of the associated column in the same numerical format as the current non-discrete axes. The numerical format of the data values can be changed through the routine ggSetAxesAnnotation() with **xory** set to which ever is the non-discrete axis.

Histogram Data Value Control

The position and orientation of each value can be changed with the routine:

```
ggSetValueAttribs(xpos,ypos,xory,xoff,yoff,angstr,justmb,juslcr)
```

where **xpos**, **ypos** and **xory** determine the position of each values control point, about which the remaining arguments refer. The control point can be positioned at one of 15 positions around each column as well as 3 positions at a specified X or Y coordinate by setting **yorx** and either **xpos** or **ypos** to GSPECIFIED. The value of **xory** is measured in graphical coordinates and so refers to the current axes ranges (as set up by ggSetAxesScaling()) for the appropriate axis.

The control points are shown below with height values drawn centrally over each one and **xory** set to 9.0.

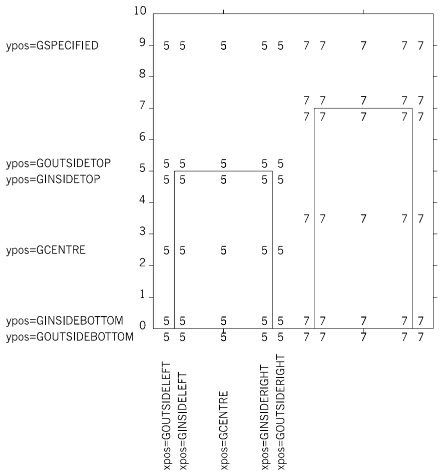


Chart annotation control positions

The routine ggSetValueAttribs() can also be used to set an additional offset (measured in user space coordinates) in the horizontal and/or vertical direction (**xoff**,**yoff**), an annotation string angle (**angstr**) and a vertical and/or horizontal justification (**justmb**,**jushor**). The default setting of centre justification is sufficient for centrally placed control positions (**xpos** = GCENTRE), but the appropriate left or right justification would usually be required for control positions set at the edges of the column limits.

Where Histogram data values are negative, the position of the control point on the non-discrete axis is placed in the matching position corresponding to the column height but below the zero axis.

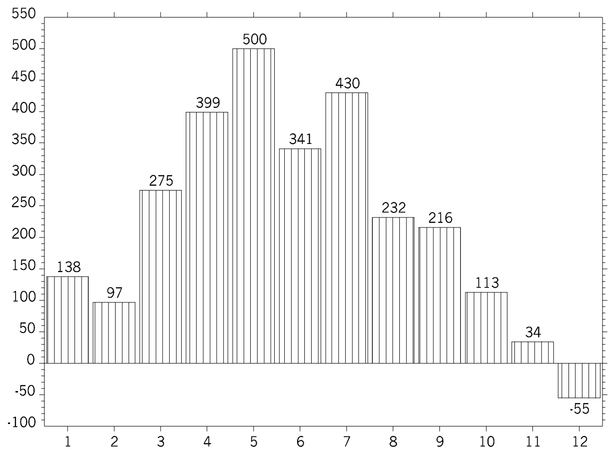
Appended to each value in any of the the value charts can be added a prefix and/or suffix string using the routine.

ggSetValueTags(prefix,suffix)

where **prefix** and **suffix** are strings of up to 30 characters that are appended to all the values in one value chart output. The prefix and/or suffix strings are included as part of the value when calculating the justified position of the total output.

Example of Fully Annotated Histogram

The following example shows the use of all the Histogram component routines:



Example of fully annotated Histogram

C Code

```
        /*  FULLY ANNOTATED HISTOGRAM */
        #include <gino-c.h>
        #include <graf-c.h>

        int main(void) {
           float yarray[12] = {138.0,97.0,275.0,399.0,
               500.0,341.0,430.0,232.0,216.0,113.0,34.0,-55.0};
           GCHASTY rep;
           GLIMIT lims;
           int i, flg, fill[12], line[12];
           for (i=0; i<12; i++) {
               fill[i]=2;
               line[i]=1;
           }

           gOpenGino();
           xxxxx();
           ggSetGraphCharMode(GGINOMODE);

        /* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
           ggEnqPlotFrame(&flg,&lims);
           gEnqCharAttribs(&rep);

        /* SET UP AXES POSITIONS AND SCALES */
           ggSetAxesPos(GAXISSTART,9.0*rep.width,
               5.0*rep.height,lims.xmax-lims.xmin-12.0*rep.width,GXAXIS);
           ggSetAxesScaling(GDISCRETE,12,1.0,12.0,GXAXIS);
           ggSetAxesPos(GAXISSTART,9.0*rep.width,
               5.0*rep.height,lims.ymax-lims.ymin-10.*rep.height,GYAXIS);
           ggSetAxesScaling(GLINEARTYPE1,12,-100.0,550.0,GYAXIS);
        /* DRAW HISTOGRAM */
           ggFillHistogram(12,yarray,0.9,fill,line);
           ggAddHistogramOutline(12,yarray,0.9);

        /* POSITION HEIGHT VALUES ABOVE COLUMNS */
           ggSetValueAttribs(GCENTRE,GOUTSIDETOP,0.0,
               0.0,0.0,0.0,GCENTRE,GCENTRE);
           ggAddHistogramValues(12,yarray,0.9);

        /* DRAW GRID */
           ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION);

           gSuspendDevice();
           gCloseGino();
           return(0);
        }
```

F90 Code

```
        ! FULLY ANNOTATED HISTOGRAM
        use gino_f90
        use graf_f90

          real, dimension(12) :: yarray = (/138.0,97.0,275.0, &
            399.0,500.0,341.0,430.0,232.0,216.0,113.0,34.0,-55.0/)
          type (GCHASTY) rep
          type (GLIMIT) lims
          integer flg
          integer, dimension(12) :: fill=(/12*2/)
          integer, dimension(12) :: line=(/12*1/)

          call gOpenGino
          call xxxxx
          call ggSetGraphCharMode(GGINOMODE)
        ! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
          call ggEnqPlotFrame(flg,lims)
          call gEnqCharAttribs(rep)

        ! SET UP AXES POSITIONS AND SCALES
          call ggSetAxesPos(GAXISSTART,9.0*rep%%width, &
               5.0*rep%height,lims%xmax-lims%xmin-12.0*rep%width,GXAXIS)
          call ggSetAxesScaling(GDISCRETE,12,1.0,12.0,GXAXIS)
          call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
               5.0*rep%height,lims%ymax-lims%ymin-10.*rep%height,GYAXIS)
          call ggSetAxesScaling(GLINEARTYPE1,12,-100.0,550.0,GYAXIS)

        ! DRAW HISTOGRAM
          call ggFillHistogram(12,yarray,0.9,fill,line)
          call ggAddHistogramOutline(12,yarray,0.9)

        ! POSITION HEIGHT VALUES ABOVE COLUMNS
          call ggSetValueAttribs(GCENTRE,GOUTSIDETOP,0.0, &
            0.0,0.0,0.0,GCENTRE,GCENTRE)
          call ggAddHistogramValues(12,yarray,0.9)

        ! DRAW GRID
          call ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Bar Chart Components

There are four components available for the building of a fully annotated, filled Bar Chart:

`ggBlockFillBarChart()`	Drawing block filled barchart
`ggAddBarChartOutline()`	Drawing bar outline
`ggFillBarChart()`	Drawing filled bars
`ggAddBarChartValues()`	Displaying start, finish or length values

All four routines may be used independently of each other or on the same chart although it is usual to use a combination of block filling and values, or simple filling, outline and values. It is necessary to define both X and Y axis positions and data ranges before calling any of these routines either by the axis definition routines or the Complete Chart Drawing routine ggPlotBarChart(). One of the axes should be defined as a discrete axis (**scale**=GDISCRETE) where upon the bars are measured against the remaining non-discrete axis.

Users should note that if ggFillBarChart() is called after ggPlotBarChart() or ggAddBarChartOutline() the Bar Chart outline will be overwritten by the filling. If the outline is required, the solution is to always draw the outline with ggAddBarChartOutline() after the filling. The block filling routine fills the appropriate areas and follows this with drawing the bar chart outline in the current GINO line colour.

Block Filled Bar Chart

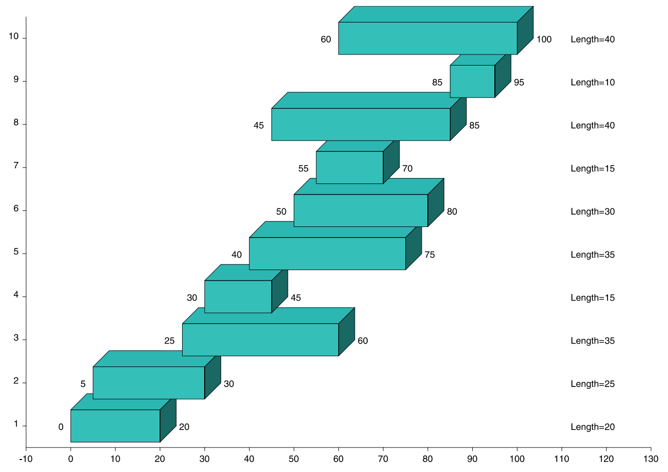
The routine to display a block filled bar chart is:

ggBlockFillBarChart(nbars,bars,frac,line)

This routine displays a set of data values as block filled bars on the last defined set of axes. The bars are plotted about the centre of the tick marks along the discrete axis according to the current block chart attributes. Each bar starts at the corresponding value of **bars.s** and finishes at the corresponding value of **bars.f**. Start and finish values may be positive or negative.

All the bars are solid filled with the specified **line** style with the extrusions filled with a darker (or lighter) shade of this line style according to the current block chart attributes (using ggSetBlockChartAttribs()).

The following shows an example of a block filled bar chart.



Block Filled Bar Chart

Bar Chart Outline

The routine to display a set of bars is:

ggAddBarChartOutline(nbars,bars,frac)

This routine displays a set of data values as bars on the last defined set of axes. The bars are plotted about the centre of the tick marks along the discrete axis in the order that they are presented within the array. Each bar starts at the corresponding value of **bars.s** and finishes at the corresponding value of **bars.f**. Start and finish values may be positive or negative.

The representation of the bar depends on the width of the bars which is determined using the parameter **frac**, the fraction of the widest possible bar given the axis length and number of bars. The effects of changing **frac** are the same as with the Histogram routine ggAddHistogramOutline().

Bar Chart Filling

The routine to display a set of filled bars is:

ggFillBarChart(nbars,bars,frac,fill,line)

This routine displays a set of data values as filled bars on the last defined set of axes. The bars are plotted about the centre of the tick marks along the discrete axis in the order that they are presented within the array.

The width of the bars is determined using the parameter **frac**, the fraction of the widest possible bar given the axis length and number of bars.

The hatch or fill style for each bar is held in the array **fill**. Various hatches and cross hatches as well as solid fill are available. If an element of **fill** contains a number less than -1 (GHOLLOW), the corresponding bar is not filled and the boundary not drawn. The line styles used to fill the bars are held in the array **line**. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. Further information on line style definition (which includes colour definition) and information on hatch and fill style definition appears in the main introduction.

The bar outlines are not drawn with this routine. Users should use ggPlotBarChart() or ggAddBarChartOutline() to draw the outline.

Annotating Bar Charts

The routine to annotate a Bar Chart is:

```
ggAddBarChartValues(nbars,bars,frac,sfl)
```

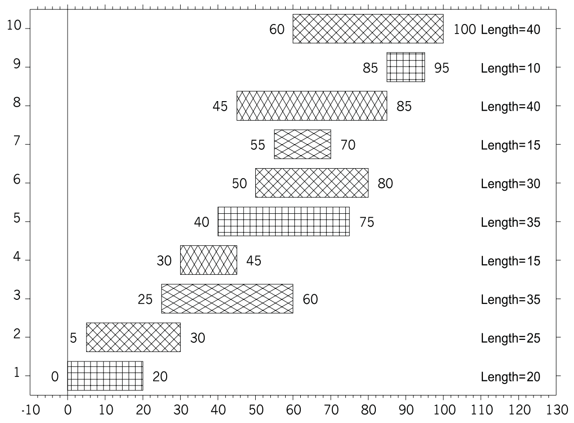
where ggAddBarChartValues() will display **nbars** values associated with the Bar Chart bars drawn by ggPlotBarChart(), ggFillBarChart() or ggAddBarChartOutline(). The argument **sfl** provides the choice of displaying start values (**sfl**=GSTART), finish values (**sfl**=GFINISH) or bar lengths (**sfl**=GLENGTH). By default, the values are positioned at the centre of the associated bar in the same numerical format as the current non-discrete axes. The numerical format of the data values can be changed through the routine ggSetAxesAnnotation() with **xory** set to whichever is the non-discrete axis.

The argument **frac** supplied to ggAddBarChartValues() is required to ensure correct positioning of the values when they are placed at the edge of each bar, it should therefore be set to the same value as that used in a corresponding call to ggPlotBarChart(), ggFillBarChart() or ggAddBarChartOutline().

Alternative positions and formatting options can be set by using the annotation control routines (see page 88). The 15 control point positions around the Bar Chart areas are located in the same logical position irrespective of the data limits or the axis direction. These routines provide for string angle and justification control as well as prefix and/or suffix strings added to each value.

Example of Fully Annotated Bar Chart

The following example shows the use of all the Bar Chart component routines:



Example of fully annotated Bar Chart

C Code

```
/*  FULLY ANNOTATED BARCHART */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   GBARCHART bars[10] = {0.0,20.0,5.0,30.0,25.0,60.0,
       30.0,45.0,40.0,75.0,50.0,80.0,55.0,70.0,45.0,85.0,
       85.0,95.0,60.0,100.0};
   GLIMIT lims;
   GCHASTY rep;
   int i, flg, line[10],
       fill[10] = {5,6,7,8,5,6,7,8,5,6};
   for (i=0; i<10; i++) line[i]=1;

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
   ggEnqPlotFrame(&flg,&lims);
   gEnqCharAttribs(&rep);

/* SET UP AXES POSITIONS AND SCALES */
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.xmax-lims.xmin-12.0*rep.width,GXAXIS);
   ggSetAxesScaling(GLINEARTYPE1,12,-10.0,130.0,GXAXIS);
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.ymax-lims.ymin-10.0*rep.height,GYAXIS);

   ggSetAxesScaling(GDISCRETE,10,1.0,10.0,GYAXIS);

/* DRAW BAR CHART */
   ggFillBarChart(10,bars,0.75,fill,line);
   ggAddBarChartOutline(10,bars,0.75);
/* POSITION START VALUE LEFT OF BAR, RIGHT JUSTIFIED */
   ggSetValueAttribs(GOUTSIDELEFT,GCENTRE,0.0,
       0.0,0.0,0.0,GCENTRE,GRIGHT);
   ggAddBarChartValues(10,bars,0.75,GSTART);

/* POSITION FINISH VALUE RIGHT OF BAR, LEFT JUSTIFIED */
   ggSetValueAttribs(GOUTSIDERIGHT,GCENTRE,0.0,
       0.0,0.0,0.0,GCENTRE,GLEFT);
   ggAddBarChartValues(10,bars,0.75,GFINISH);

/* POSITION LENGTH VALUE AT 110.0 LEFT JUSTIFIED */
   ggSetValueAttribs(GSPECIFIED,GCENTRE,110.0,
       0.0,0.0,0.0,GCENTRE,GLEFT);
   ggSetValueTags("Length=”," ");
   ggAddBarChartValues(10,bars,0.75,GLENGTH);

/* DRAW GRID */
   ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 Code

```
! FULLY ANNOTATED BARCHART
use gino_f90
use graf_f90

  type (GBARCHART), dimension(10) :: bars = &
    (/GBARCHART(0.0,20.0),GBARCHART(5.0,30.0), &
      GBARCHART(25.0,60.0),GBARCHART(30.0,45.0), &
      GBARCHART(40.0,75.0),GBARCHART(50.0,80.0), &
      GBARCHART(55.0,70.0),GBARCHART(45.0,85.0), &
      GBARCHART(85.0,95.0),GBARCHART(60.0,100.0)/)
  type (GLIMIT) lims
  type (GCHASTY) rep
  integer i, flg
  integer, dimension(10) :: line = (/10*1/)
  integer, dimension(10) :: fill = (/5,6,7,8,5,6,7,8,5,6/)

  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

! ENQUIRE GINOGRAF DRAWING LIMITS
! AND CHARACTER SIZE
  call ggEnqPlotFrame(flg,lims)
  call gEnqCharAttribs(rep)

! SET UP AXES POSITIONS AND SCALES
  call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
       5.0*rep%height,lims%xmax-lims%xmin-12.0*rep%width,GXAXIS)
  call ggSetAxesScaling(GLINEARTYPE1,12,-10.0,130.0,GXAXIS)
  call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
       5.0*rep%height,lims%ymax-lims%ymin-10.0*rep%height,GYAXIS)
  call ggSetAxesScaling(GDISCRETE,10,1.0,10.0,GYAXIS)
```

```
        ! DRAW BAR CHART
          call ggFillBarChart(10,bars,0.75,fill,line)
          call ggAddBarChartOutline(10,bars,0.75)

        ! POSITION START VALUE LEFT OF BAR, RIGHT JUSTIFIED
          call ggSetValueAttribs(GOUTSIDELEFT,GCENTRE,0.0,  &
            0.0,0.0,0.0,GCENTRE,GRIGHT)
          call ggAddBarChartValues(10,bars,0.75,GSTART)

        ! POSITION FINISH VALUE RIGHT OF BAR, LEFT JUSTIFIED
          call ggSetValueAttribs(GOUTSIDERIGHT,GCENTRE,0.0,  &
            0.0,0.0,0.0,GCENTRE,GLEFT)
          call ggAddBarChartValues(10,bars,0.75,GFINISH)

        ! POSITION LENGTH VALUE AT 110.0 LEFT JUSTIFIED
          call ggSetValueAttribs(GSPECIFIED,GCENTRE,110.0,  &
            0.0,0.0,0.0,GCENTRE,GLEFT)
          call ggSetValueTags('Length=',' ')
          call ggAddBarChartValues(10,bars,0.75,GLENGTH)

        ! DRAW GRID */
          call ggAddGrid(GINTERMEDIATE,GTICKS,GANNOTATION,GANNOTATION)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Step Chart Components

There are four components available for the building of a fully annotated, filled Step Chart:

`ggBlockFillStepChart()`	Drawing block filled step chart
`ggAddStepChartOutline()`	Drawing step or column outline
`ggFillStepChart()`	Drawing filled steps
`ggAddStepChartValues()`	Displaying step values

All four routines may be used independently of each other or on the same chart although it is usual to use a combination of block filling and values, or simple filling, outline and values. It is necessary to define both X and Y axis positions and data ranges before calling any of these routines either by the axis definition routines or the Complete Chart Drawing routine ggPlotStepChart().

Users should note that if ggFillStepChart() is called after ggPlotStepChart() or ggAddStepChartOutline() the Step Chart outline will be overwritten by the filling. If the outline is required, the solution is to always draw the outline with ggAddStepChartOutline() after the filling. The block filling routine fills the appropriate areas and follows this with drawing the bar chart outline in the current GINO line colour.

Block Filled Step Chart

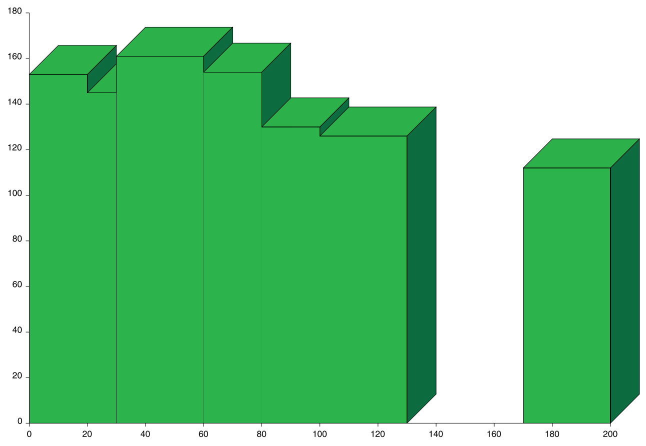
The routine to display a block filled step chart is:

ggBlockFillStepChart(nsteps,steps,base,xory,line)

This routine displays a set of data values as block filled columns on the last defined set of axes. The columns are of various widths being between the values supplied in the array **steps**. Each column is filled between the height value in the same array and the constant value **base**. The argument **xory** determines whether the columns are oriented with the heights against the vertical (**xory**=GXAXIS) or horizontal (**xory**=GYAXIS) axis.

All the blocks are solid filled with the specified **line** style with the extrusions filled with a darker (or lighter) shade of this line style according to the current block chart attributes (using ggSetBlockChartAttribs()).

The following shows an example of a block filled step chart.



Block Filled Step Chart

Step Chart Outline

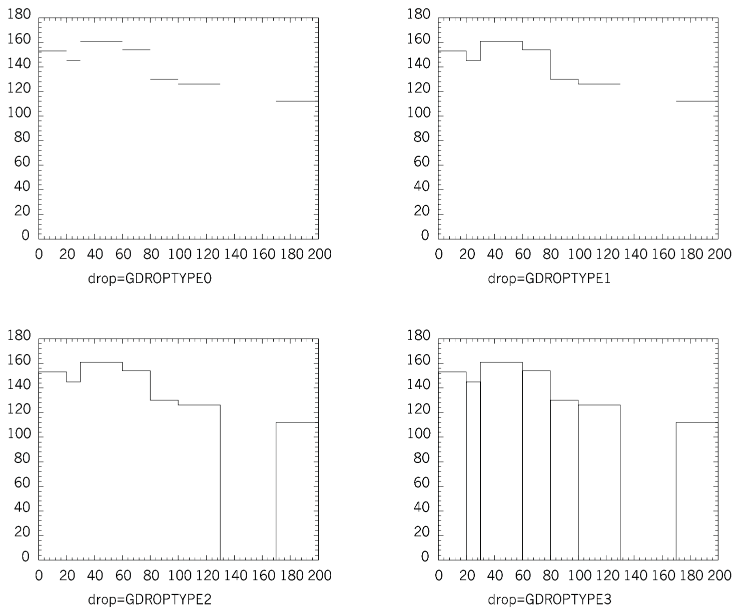
The routine to display a set of steps is:

ggAddStepChartOutline(nsteps,steps,base,drop,xory)

This routine displays a set of data values as steps or columns on the last defined set of axes. The start, finish and height values of each step are held in the array **steps** (of type GSTEPCHART). The argument **xory** determines whether the heights are plotted against the vertical (**xory**=GXAXIS) or horizontal (**xory**=GYAXIS) axis.

An optional **base** value can also be supplied where the steps are required to descend to a fixed value. The use of **base** is in conjunction with the argument **drop** which determines the form of the Step Chart display such that where **drop**=GDROPTYPE0 or GDROPTYPE1, **base** is not required, but where **drop**=GDROPTYPE2 or GDROPTYPE3, **base** is required.

The four Step Chart forms are shown below:



Four different Step Chart forms

Step Chart Filling

The routine to display a set of filled steps is:

```
ggFillStepChart(nsteps,steps,base,xory,fill,line)
```

This routine displays a set of data values as filled columns on the last defined set of axes. The columns are of various widths being between the values supplied in the array **steps** (of type GSTEPCHART). Each column is filled between the height value in the same array and the constant value **base**. The argument **xory** determines whether the columns are oriented with the heights against the vertical (**xory**=GXAXIS) or horizontal (**xory**=GYAXIS) axis. When using ggFillStepChart() in conjunction with ggPlotStepChart(), **xory** should be set to GXAXIS.

The hatch or fill style for each column is held in the array **fill**. Various hatches and cross hatches as well as solid fill are available. If an element of **fill** contains a number less than -1 (GHOLLOW), the corresponding column is not filled and the boundary not drawn. The line styles used to fill the columns are held in the array **line**. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. Further information on line style definition (which includes colour definition) and information on hatch and fill style definition appears in the main introduction.

The step outlines are not drawn with this routine. Users should use ggPlotStepChart() or ggAddStepChartOutline() to draw the outline.

Annotating Step Charts

The routine to annotate a Step Chart is:

ggAddStepChartValues(nsteps,steps,base,sfl,xory)

where ggAddStepChartValues() will display **nsteps** values associated with the Step Chart drawn by ggPlotStepChart() or ggAddStepChartOutline(). The argument **sfl** provides the choice of displaying five different values:

sfl=	`GSTART`	start value of each step
	`GFINISH`	finish value of each step
	`GHEIGHT`	height value of each step
	`GWIDTH`	width of each step
	`GHEIGHTABOVEBASE`	height above base of each step

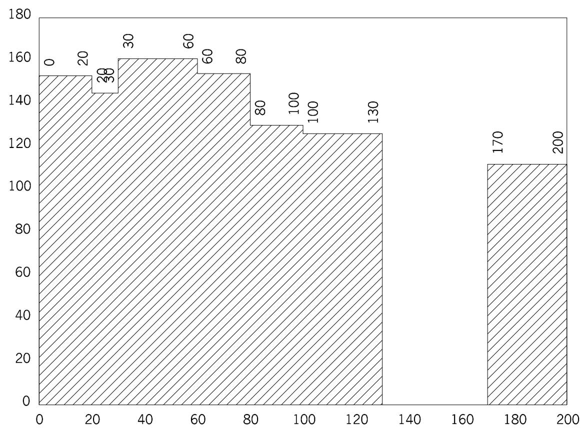
By default, the values are positioned at the centre of the associated column (ie, between the height in **steps** and the **base** value) in the same numerical format as their associated axis. The numerical format of the data values can be changed through the routine ggSetAxesAnnotation() with **xory** set appropriately.

The user can select alternative positions and formatting options by using the annotation control routines (see page 88). The 15 control point positions around the Step Chart areas are located in the same logical position irrespective of the data limits or the axis direction. These routines also provide for string angle and justification control as well as prefix and/or suffix strings added to each value.

Although the step outline is not drawn with this routine is it necessary to set the correct orientation of the columns to ensure the values are correctly positioned. This is determined by the argument **xory** such that when **xory**=GXAXIS the height of the column is oriented against the vertical axis and when **xory**=GYAXIS the height of the column is oriented against the horizontal axis. When using ggAddStepChartValues() in conjunction with ggPlotStepChart(), **xory** should be set to GXAXIS.

Example of Fully Annotated Step Chart

The following example shows the use of all the Step Chart component routines:



Example of fully annotated Step Chart

C Code

```
/* FULLY ANNOTATED STEP CHART */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   GSTEPCHART steps[7] = {0.0,20.0,153.0,20.0,30.0,145.0,
       30.0,60.0,161.0,60.0,80.0,154.0,80.0,100.0,130.0,
       100.0,130.0,126.0,170.0,200.0,112.0};
   GCHASTY rep;
   GLIMIT lims;
   int i, flg, fill[7], line[7];
   for (i=0; i<7; i++) {
        fill[i]=4;
        line[i]=1;
   }

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
   ggEnqPlotFrame(&flg,&lims);
   gEnqCharAttribs(&rep);

/* SET UP AXES POSITIONS AND SCALES */
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.xmax-lims.xmin-12.0*rep.width,GXAXIS);
  ggSetAxesScaling(GLINEARTYPE1,11,0.0,200.0,GXAXIS);
  ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.ymax-lims.ymin-10.0*rep.height,GYAXIS);

   ggSetAxesScaling(GLINEARTYPE1,10,0.0,180.0,GYAXIS);

/* DRAW STEP CHART */
   ggFillStepChart(7,steps,0.0,GXAXIS,fill,line);
   ggAddStepChartOutline(7,steps,0.0,GDROPTYPE3,GXAXIS);

/* POSITION START VALUES AT LEFT EDGE OF COLUMN AT 90 DEGREES */
   ggSetValueAttribs(GINSIDELEFT,GOUTSIDETOP,
       0.0,0.0,0.0,90.0,GCENTRE,GLEFT);
   ggAddStepChartValues(7,steps,0.0,GSTART,GXAXIS);

/* POSITION FINISH VALUES AT RIGHT EDGE OF COLUMN AT 90 DEGREES */
   ggSetValueAttribs(GINSIDERIGHT,GOUTSIDETOP,
       0.0,0.0,0.0,90.0,GCENTRE,GLEFT);
   ggAddStepChartValues(7,steps,0.0,GFINISH,GXAXIS);

/* DRAW GRID */
   ggAddGrid(GNONE,GNONE,GANNOTATION,GANNOTATION);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 Code

```
! FULLY ANNOTATED STEP CHART
use gino_f90
use graf_f90

  type (GSTEPCHART), dimension(7) :: steps = &
    (/GSTEPCHART(0.0,20.0,153.0),GSTEPCHART(20.0,30.0,145.0), &
      GSTEPCHART(30.0,60.0,161.0), &
      GSTEPCHART(60.0,80.0,154.0), &
      GSTEPCHART(80.0,100.0,130.0), &
      GSTEPCHART(100.0,130.0,126.0), &
      GSTEPCHART(170.0,200.0,112.0)/)
  type (GCHASTY) rep
  type (GLIMIT) lims
  integer i, flg
  integer, dimension(7) :: fill = (/7*4/)
  integer, dimension(7) :: line = (/7*1/)

  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
  call ggEnqPlotFrame(flg,lims)
  call gEnqCharAttribs(rep)

! SET UP AXES POSITIONS AND SCALES
  call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
       5.0*rep%height,lims%xmax-lims%xmin-12.0*rep%width,GXAXIS)
  call ggSetAxesScaling(GLINEARTYPE1,11,0.0,200.0,GXAXIS)
  call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
       5.0*rep%height,lims%ymax-lims%ymin-10.0*rep%height,GYAXIS)
  call ggSetAxesScaling(GLINEARTYPE1,10,0.0,180.0,GYAXIS)

! DRAW STEP CHART
  call ggFillStepChart(7,steps,0.0,GXAXIS,fill,line)
  call ggAddStepChartOutline(7,steps,0.0,GDROPTYPE3,GXAXIS)

! POSITION START VALUES AT LEFT EDGE OF COLUMN AT 90 DEGREES
  call ggSetValueAttribs(GINSIDELEFT,GOUTSIDETOP,
       0.0,0.0,0.0,90.0,GCENTRE,GLEFT)
  call ggAddStepChartValues(7,steps,0.0,GSTART,GXAXIS)

! POSITION FINISH VALUES AT RIGHT EDGE OF COLUMN AT 90 DEGREES
  call ggSetValueAttribs(GINSIDERIGHT,GOUTSIDETOP, &
       0.0,0.0,0.0,90.0,GCENTRE,GLEFT)
  call ggAddStepChartValues(7,steps,0.0,GFINISH,GXAXIS)

! DRAW GRID
  call ggAddGrid(GNONE,GNONE,GANNOTATION,GANNOTATION)

  call gSuspendDevice
  call gCloseGino
  stop
  end
```

Area Chart Components

There are four components available for the building of a fully annotated, filled Area Chart:

`ggBlockFillAreaChart()`	Drawing block filled area chart
`ggAddAreaChartOutline()`	Drawing area or column outline
`ggFillAreaChart()`	Drawing filled areas
`ggAddAreaChartValues()`	Displaying area values

All four routines may be used independently of each other or on the same chart although it is usual to use a combination of block filling and values, or simple filling, outline and values. It is necessary to define both X and Y axis positions and data ranges before calling any of these routines either by the axis definition routines or the Complete Chart Drawing routine ggPlotAreaChart().

Users should note that if ggFillAreaChart() is called after ggPlotAreaChart() or ggAddAreaChartOutline() the Area Chart outline will be overwritten by the filling. If the outline is required, the solution is to always draw the outline with ggAddAreaChartOutline() after the filling. The block filling routine fills the appropriate areas and follows this with drawing the area chart outline in the current GINO line colour.

Block Filled Area Chart

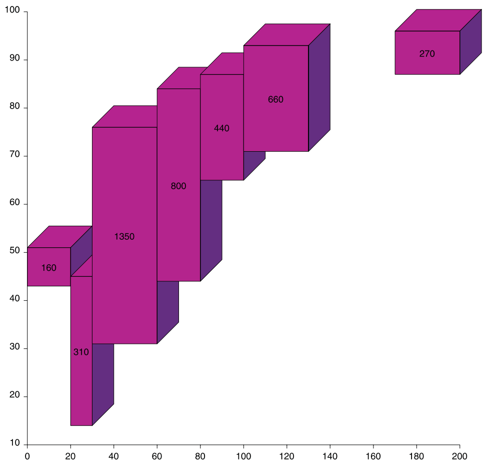
The routine to display a block filled area chart is:

ggBlockFillAreaChart(nareas,areas,xory,line)

This routine displays a set of data values as filled areas on the last defined set of axes. The width of each area is defined to be between the values supplied in the elements **areas.s** and **areas.f** and the height of each area is defined to be between the values supplied in the elements **areas.h1** and **areas.h2**. The argument **xory** determines whether the areas are oriented with the heights against the vertical (**xory**=GXAXIS) or horizontal (**xory**=GYAXIS) axis.

All the blocks are solid filled with the specified **line** style with the extrusions filled with a darker (or lighter) shade of this line style according to the current block chart attributes (using ggSetBlockChartAttribs()).

The following shows an example of a block filled area chart.



Block Filled Area Chart

Area Chart Outline

The routine to display a set of areas is:

ggAddAreaChartOutline(nareas,areas,xory)

This routine displays a set of data values as areas on the last defined set of axes. The start and finish values of each area are held in the elements **areas.s** and **areas.f**, with the lower and upper values in the corresponding elements of the **areas.h1** and **areas.h2**. The argument **xory** determines whether the heights are plotted against the vertical (**xory**=GXAXIS) or horizontal (**xory**=GYAXIS) axis.

Area Chart Filling

The routine to display a set of filled areas is:

ggFillAreaChart(nareas,areas,xory,fill,line)

This routine displays a set of data values as filled areas on the last defined set of axes. The width of each area is defined to be between the values supplied in the elements **areas.s** and **areas.f** and the height of each area is defined to be between the values supplied in the elements **areas.h1** and **areas.h2**. The argument **xory** determines whether the areas are oriented with the heights against the vertical (**xory**=GXAXIS) or horizontal (**xory**=GYAXIS) axis. When using ggFillAreaChart() in conjunction with ggPlotAreaChart(), **xory** should be set to GXAXIS.

The hatch or fill style for each area is held in the array **fill**. Various hatches and cross hatches as well as solid fill are available. If an element of **fill** contains a number less than -1 (GHOLLOW), the corresponding area is not filled and the boundary not drawn. The line styles used to fill the areas are held in the array **line**. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. Further information on line style definition (which includes colour definition) and information on hatch and fill style definition appears in the main introduction.

The area outlines are not drawn with this routine. Users should use ggPlotAreaChart() or ggAddAreaChartOutline() to draw the outline.

Annotating Area Charts

The routine to annotate an Area Chart is:

```
ggAddAreaChartValues(nareas,areas,sfl,xory)
```

where ggAddAreaChartValues() will display **nareas** values associated with the Area Chart drawn by ggPlotAreaChart() or ggAddAreaChartOutline(). The argument **sfl** provides the choice of displaying seven different values:

sfl =	`GSTART`	start value of each area
	`GFINISH`	finish value of each area
	`GLOWER`	lower height value of each area
	`GUPPER`	upper height value of each area
	`GWIDTH`	width of each area
	`GHEIGHT`	height of each area
	`GAREA`	area of each area

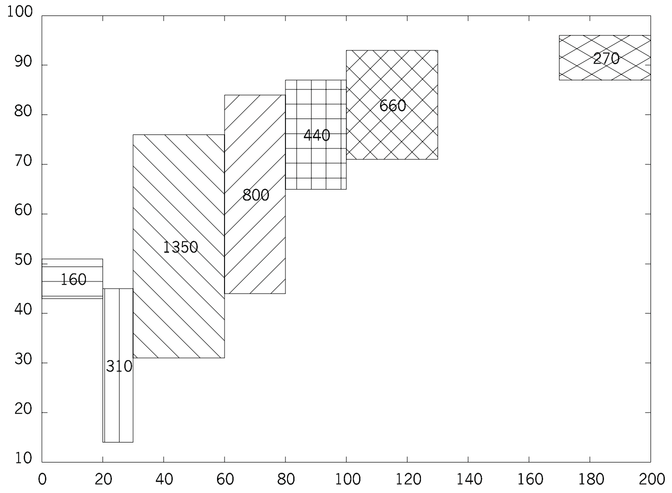
By default, the values are positioned at the centre of the associated area in the same numerical format as their associated axis. The numerical format of the data values can be changed through the routine ggSetAxesAnnotation() with **xory** set appropriately.

The user can select alternative positions and formatting options by using the annotation control routines (see page 88). The 15 control point positions around the Area Chart areas are located in the same logical position irrespective of the data limits or the axis direction. These routines also provide for string angle and justification control as well as prefix and/or suffix strings added to each value.

Although the area outline is not drawn with this routine it is necessary to set the correct orientation of the areas to ensure the values are correctly positioned. This is determined by the argument **xory** such that when **xory**=GXAXIS the height of the area is oriented against the vertical axis and when **xory**=GYAXIS the height of the area is oriented against the horizontal axis. When using ggAddAreaChartValues() in conjunction with ggPlotAreaChart(), **xory** should be set to GXAXIS.

Example of Fully Annotated Area Chart

The following example shows the use of all the Area Chart component routines:



Example of fully annotated Area Chart

C Code

```
/*  FULLY ANNOTATED AREA CHART */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   GAREACHART areas[7]={0.0,20.0,43.0,51.0,
       20.0,30.0,14.0,45.0,30.0,60.0,31.0,76.0,
       60.0,80.0,44.0,84.0,80.0,100.0,65.0,87.0,
       100.0,130.0,71.0,93.0,170.0,200.0,87.0,96.0};

   GCHASTY rep;
   GLIMIT lims;
   int i, flg, fill[7]={9,10,11,12,13,14,15},line[7];
   for (i=0; i<7; i++) line[i]=1;

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
   ggEnqPlotFrame(&flg,&lims);
   gEnqCharAttribs(&rep);

/* SET UP AXES POSITIONS AND SCALES */
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.xmax-lims.xmin-12.0*rep.width,GXAXIS);
   ggSetAxesScaling(GLINEARTYPE1,11,0.0,200.0,GXAXIS);
   ggSetAxesPos(GAXISSTART,9.0*rep.width,
       5.0*rep.height,lims.ymax-lims.ymin-10.0*rep.height,GXAXIS);

   ggSetAxesScaling(GLINEARTYPE1,10,10.0,100.0,GYAXIS);

/* DRAW AREA CHART */
   ggFillAreaChart(7,areas,GXAXIS,fill,line);
   ggAddAreaChartOutline(7,areas,GXAXIS);

/* LEAVE DEFAULT CENTRAL POSITION OF AREA VALUES */
   ggAddAreaChartValues(7,areas,GAREA,GXAXIS);

/* DRAW GRID */
   ggAddGrid(GCARDINAL,GTICKS,GANNOTATION,GANNOTATION);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 Code

```
        ! FULLY ANNOTATED AREA CHART
        use gino_f90
        use graf_f90


          type (GAREACHART), dimension(7) :: areas= &
            (/GAREACHART(0.0,20.0,43.0,51.0), &
              GAREACHART(20.0,30.0,14.0,45.0), &
              GAREACHART(30.0,60.0,31.0,76.0), &
              GAREACHART(60.0,80.0,44.0,84.0), &
              GAREACHART(80.0,100.0,65.0,87.0), &
              GAREACHART(100.0,130.0,71.0,93.0), &
              GAREACHART(170.0,200.0,87.0,96.0)/)
          type (GCHASTY) rep
          type (GLIMIT) lims
          integer i, flg
          integer, dimension(7) :: fill=(/9,10,11,12,13,14,15/)
          integer, dimension(7) :: line=(/7*1/)

          call gOpenGino
          call xxxxx
          call ggSetGraphCharMode(GGINOMODE)

        ! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
          call ggEnqPlotFrame(flg,lims)
          call gEnqCharAttribs(rep)

        ! SET UP AXES POSITIONS AND SCALES
          call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
               5.0*rep%height,lims%xmax-lims%xmin-12.0*rep%width,GXAXIS)
          call ggSetAxesScaling(GLINEARTYPE1,11,0.0,200.0,GXAXIS)
          call ggSetAxesPos(GAXISSTART,9.0*rep%width, &
               5.0*rep%height,lims%ymax-lims%ymin-10.0*rep%height,GXAXIS)
          call ggSetAxesScaling(GLINEARTYPE1,10,10.0,100.0,GYAXIS)

        ! DRAW AREA CHART
          call ggFillAreaChart(7,areas,GXAXIS,fill,line)
          call ggAddAreaChartOutline(7,areas,GXAXIS)

        ! LEAVE DEFAULT CENTRAL POSITION OF AREA VALUES
          call ggAddAreaChartValues(7,areas,GAREA,GXAXIS)

        ! DRAW GRID
          call ggAddGrid(GCARDINAL,GTICKS,GANNOTATION,GANNOTATION)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Multi Data Set Histogram Components

Multi data set histogram component routines can be used to display stacked or clustered column charts representing a grid of data from a two dimensional data array. Any contiguous data block can be extracted where the first dimension represents the number of columns and the second dimension represents the number of data sets.

There are two components available for the drawing of a stacked or clustered histograms:

`ggBlockFillMultiHistogram()`	Drawing block filled columns
`ggFillMultiHistogram()`	Drawing filled columns

In both cases it is necessary to define both X and Y axis positions and data ranges before calling any of these routines using the axis definition routines. One of the axes should be defined as a discrete axis (**scale**=GDISCRETE) whereupon the heights are measured against the remaining non-discrete axis.

Users should note that the block filling routine fills the appropriate areas and follows this with drawing the column outline in the current GINO line colour.

Multi Data Set Block Filled Histogram

The routine to display a block filled, multi-data set histogram is:

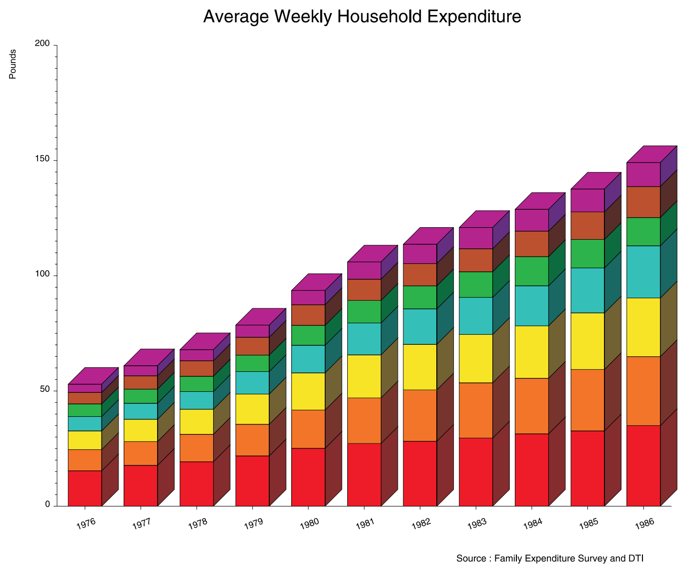
```
ggBlockFillMultiHistogram(type,rdata,ndim1,ncols,ndata,frac,gap,
          line,is1,is2)
```

Where **type** can be GSTACKED or GCLUSTERED depending on the type of display required. The data array **rdata** should contain **ndata** sets each with **ncols** of data, with an optional offset from the start of the array set by **is1**,**is2**. The argument **ndim1** represents the primary dimension of the two dimensional array.

The argument **frac** determines the width of the stacked/clustered column and **gap** determines the gap between each clustered column for **type** = GCLUSTERED.

The columns of each data set [i] are solid filled with the line style specified in **line[i]** with the extrusions filled with a darker (or lighter) shade of this line style according to the current block chart attributes (using ggSetBlockChartAttribs()).

The following shows an example of a block filled multi-data set histogram.



Block Filled Multi-Data Set Histogram

C Code

```
#include <gino-c.h>
#include <graf-c.h>

#define NSET 7 /* Number of data sets */
#define NDATA 11 /* Number of data components */

#define MIN(x,y)((x) < (y) ? (x) : (y))

/* Date Data */
char  *dates[] = {"1976","1977","1978","1979","1980","1981",
                  "1982","1983","1984","1985","1986"};

/*  data to be plotted. */
float rdata[][NDATA] = {
{15.36,17.74,19.31,21.83,25.15,27.2, 28.19,29.56,31.43,32.7, 34.97},
{ 9.21,10.31,11.87,13.72,16.56,19.76,22.29,23.99,24.06,26.63,29.92},
{ 8.14, 9.71,10.9, 13.13,16.15,18.7, 19.79,20.96,22.77,24.56,25.43},
{ 6.19, 6.93, 7.66, 9.74,11.96,13.84,15.37,16.09,17.41,19.48,22.67},
{ 5.5,  6.11, 6.64, 7.14, 8.66, 9.8,  9.98,11.12,12.62,12.38,12.26},
{ 4.99, 5.78, 6.78, 7.79, 8.99, 9.23, 9.69,10.0, 11.1, 11.92,13.46},
{3.53, 4.38, 4.76, 5.25, 6.15, 7.46, 8.35, 9.22, 9.42, 9.95,10.43}};

/* Filling and line styles */
int fill[NSET] = {GSOLID,GSOLID,GSOLID,GSOLID,GSOLID,GSOLID,GSOLID};
int line[NSET]= {GRED,GORANGE,GYELLOW,GCYAN,GGREEN,GBROWN,GMAGENTA};

/* title */
```

```
         char title1[] = "Average Weekly Household Expenditure";

         GDIM paper;

         int main ()
         {
            int ipapty;
            float factor,cw,ch,xlen,ylen,xgap,ygap;
            float ffrac=0.9,gap=0.0;
            int coloff;
            float azim,elev,depth,top,side;
         /* Enter GINO & initialise device. */

            gOpenGino();
         /* Nominate the device */
           xxxxx();

         /* Scale output to paper size. */

            gEnqDrawingLimits(&paper, &ipapty);
            factor = MIN(paper.xpap/1000.0, paper.ypap/750.0);
            gDefinePictureUnits(factor);
            gNewDrawing();

            ggRestoreAxesSettings();
            cw = 7.5;
            ch = 7.5;
            gSetCharSize(cw, ch);

         /* Set up size of graph. */

            xlen = 1000.0*2.0/3.0;
            ylen = 750.0*2.0/3.0;
            xgap = xlen/4.0;
            ygap = ylen/4.0;

         /* Position & scale the X-axis. */

            ggSetAxesPos(GAXISSTART, xgap, ygap, xlen, GXAXIS);
            ggSetAxesScaling(GDISCRETE, NDATA, 1.0, 11.0, GXAXIS);

         /* Position & scale the Y-axis. */

            ggSetAxesPos(GAXISSTART, xgap, ygap, ylen, GYAXIS);
            ggSetAxesScaling(GLINEARTYPE3, 4, 0.0, 200.0, GYAXIS);

         /* Plot data sets. */
            ggBlockFillMultiHistogram(GSTACKED,(float *)rdata,NDATA,
                   NDATA,NSET,ffrac,gap,line,1,1);

         /* Draw & title X-axis. */

            ggSetAxesAttribs(GOFFSET, ygap-2.0*ch, 1, -1, 0.0, 20.0,
               GDEFAULTPOSITION, GDEFAULTPOSITION, GNOREDUCE, GXAXIS);
            ggDrawAxes(GCARDINAL, GCLOCKWISE, GNOVAL, GXAXIS);
            ggDrawAxesLabels(NDATA, dates, GCLOCKWISE, GXAXIS);

         /* Draw & title Y-axis. */

            ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE,GANTICLOCKWISE,GYAXIS);
            ggDrawAxesTitle("Pounds", xgap-6.0*cw, GYAXIS, GLEFT, GTOP);

         /* Write title. */

            gSetCharSize(2.0*ch, 2.0*cw);
         ggDrawAxesTitle(title1, 650.0, GXAXIS, GBOTTOM, GCENTRE);

      /* Credit. */

         gSetCharSize(8.0, 8.0);
         gMoveTo2D(600.0, 65.0);
         gDisplayStr("Source : Family Expenditure Survey and DTI");

      /* Close down device & leave GINO. */

         gSuspendDevice();
         gCloseGino();
      }
```

F90 Code

```
      use gino f90
      use graf f90
      parameter (NSET=7,NDATA=11)

      ! Date Data
      character (len=4) ,dimension(NDATA) :: dates = (/ &
         '1976','1977','1978','1979','1980','1981', &
         '1982','1983','1984','1985','1986'/)

      !  data to be plotted.
      real,dimension(NDATA,NSET) :: data = reshape( (/ &
      15.36,17.74,19.31,21.83,25.15,27.2, 28.19,29.56 31.43,32.7, 34.97, &
       9.21,10.31,11.87,13.72,16.56,19.76,22.29,23.99,24.06,26.63,29.92, &
       8.14, 9.71,10.9, 13.13,16.15,18.7, 19.79,20.96,22.77,4.56, 25.43, &
       6.19, 6.93, 7.66, 9.74,11.96,13.84,15.37,16.09,17.41,19.48,22.67, &
       5.5,  6.11, 6.64, 7.14, 8.66, 9.8,  9.98,11.12,12.62,12.38,12.26, &
       4.99, 5.78, 6.78, 7.79, 8.99, 9.23, 9.69,10.0, 11.1, 11.92,13.46, &
       3.53, 4.38, 4.76, 5.25, 6.15, 7.46, 8.35, 9.22, 9.42, 9.95,10.43  &
       /) , (/NDATA,NSET/) )

      ! Filling and line styles
      integer, dimension(NSET) :: fill = &
          (/GSOLID,GSOLID,GSOLID,GSOLID,GSOLID,GSOLID,GSOLID/)
      integer, dimension(NSET) :: line = &
          (/GRED,GORANGE,GYELLOW,GCYAN,GGREEN,GBROWN,GMAGENTA/)

      !
      ! title
      !
      character (len=41) :: title1 = &
           'Average Weekly Household Expenditure'

      type (GDIM) paper

      ! Enter GINO & initialise device.

         call gOpenGino
         call xxxxx
      !
      ! Scale output to paper size.
      !
         call gEnqDrawingLimits(paper, ipapty)
         factor = MIN(paper%xpap/1000.0, paper%ypap/750.0)
         call gDefinePictureUnits(factor)
         call gNewDrawing

         call ggRestoreAxesSettings
      cw = 7.5
      ch = 7.5
      call gSetCharSize(cw, ch)
     !
     ! Set up size of graph.
     !
      xlen = 1000.0*2.0/3.0
      ylen = 750.0*2.0/3.0
      xgap = xlen/4.0
      ygap = ylen/4.0
     !
     ! Position & scale the X-axis
     !
      call ggSetAxesPos(GAXISSTART, xgap, ygap, xlen, GXAXIS)
      call ggSetAxesScaling(GDISCRETE, NDATA, 1.0, 11.0, GXAXIS)
     !
     ! Position & scale the Y-axis
     !
      call ggSetAxesPos(GAXISSTART, xgap, ygap, ylen, GYAXIS)
      call ggSetAxesScaling(GLINEARTYPE3, 4, 0.0, 200.0, GYAXIS)
     !
     ! Plot data sets
     !
      call ggBlockFillMultiHistogram(GSTACKED,data,NDATA, &
                                NDATA,NSET,0.9,0.0,line,1,1)
     !
     ! Draw & title X-axis.
     !
      call ggSetAxesAttribs(GOFFSET, ygap-2.0*ch, 1, -1, 0.0, 20.0, &
         GDEFAULTPOSITION, GDEFAULTPOSITION, GNOREDUCE, GXAXIS)
      call ggDrawAxes(GCARDINAL, GCLOCKWISE, GNOVAL, GXAXIS)
      call ggDrawAxesLabels(NDATA, dates, GCLOCKWISE, GXAXIS)
     !
     ! Draw & title Y-axis.
     !
      call ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE,GANTICLOCKWISE, &
                  GYAXIS)
      call ggDrawAxesTitle('Pounds', xgap-6.0*cw, GYAXIS, GLEFT, GTOP)
     !
     ! Write title.
     !
      call gSetCharSize(2.0*ch, 2.0*cw)
      call ggDrawAxesTitle(title1, 650.0, GXAXIS, GBOTTOM, GCENTRE)
     !
     ! Credit.
     !
      call gSetCharSize(8.0, 8.0)
      call gMoveTo2D(600.0, 65.0)
      call gDisplayStr('Source : Family Expenditure Survey and DTI')
     !
     ! Close down device & leave GINO.
     !
      call gSuspendDevice()
      call gCloseGino()
      stop
      end
```

Multi Data Set Histogram Filling

The routine to display a filled, multi-data set histogram is:

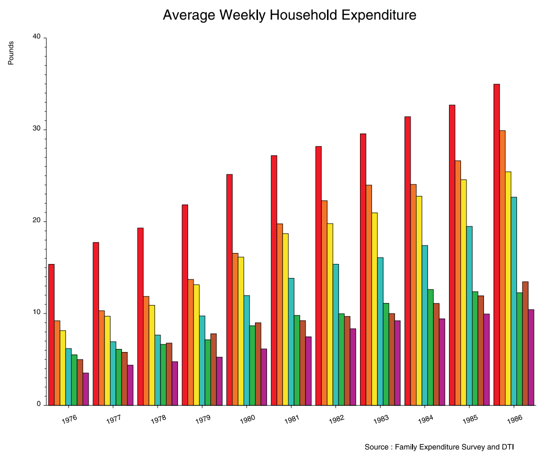
ggFillMultiHistogram(type,rdata,ndim1,ncols,ndata,frac,gap,fill,line,is1,is2)

Where **type** can be GSTACKED or GCLUSTERED depending on the type of display required. The data array **rdata** should contain **ndata** sets each with **ncols** of data, with an optional offset from the start of the array set by **is1**,**is2**. The argument **ndim1** represents the primary dimension of the two dimensional array.

The argument **frac** determines the width of the stacked/clustered column and **gap** determines the gap between each clustered column for **type** = GCLUSTERED.

The columns of each data set [i] are filled with fill style **fill[i]** and line style **line[i]**. Various hatches and cross hatches as well as solid fill are available. If an element of **fill** contains a number less than -1(GHOLLOW), the corresponding column is not filled and the boundary not drawn. The line styles used to fill the columns are held in the array **line**. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. Further information on line style definition (which includes colour definition) and information on hatch and fill style definition appears in the main introduction.

N.B. Note that the outline of the columns drawn by ggFillMultiHistogram() DO NOT match those drawn by ggBlockFillMultiHistogram() due to the extra width required by the extrusions in the latter routine.



Filled Multi-Data Set Histogram

The above chart can be displayed with similar code to the block filled example shown above with the following changes:

C Code

```
/* Position & scale the Y-axis. */

   ggSetAxesPos(GAXISSTART, xgap, ygap, ylen, GYAXIS);
   ggSetAxesScaling(GLINEARTYPE3, 4, 0.0, 40.0, GYAXIS);

/* Plot data sets. */
   ggFillMultiHistogram(GCLUSTERED,(float *)rdata,NDATA,
              NDATA,NSET,ffrac,gap,fill,line,1,1);
```

F90 Code

```
!
! Position & scale the Y-axis.
!
   call ggSetAxesPos(GAXISSTART, xgap, ygap, ylen, GYAXIS)
   call ggSetAxesScaling(GLINEARTYPE3, 4, 0.0, 40.0, GYAXIS)
!
! Plot data sets.
!
   call ggFillMultiHistogram(GCLUSTERED,data,NDATA, &
              NDATA,NSET,0.9,0.0,fill,line,1,1)
```

Chapter 5

VECTOR CHARTS

Introduction to Vector Charts

This chapter describes a special form of chart containing vectors (or scalars) representing a grid of directions, strengths and colours providing a means of displaying 5 sets of information on one chart. Vector Charts are drawn with reference to the current graphical axes system as set up with the axis definition routines. Facilities are provided to map the Vector Chart onto different areas within the axis coordinate system and to clip and scale the vectors that are drawn. There is no complete Vector Chart routine as with the previous chart types.

Summary of Vector Chart facilities:

`ggSetVectorChartFrame()`
`ggEnqVectorChartFrame()`
	Set/enquire boundary limits of Vector Chart

`ggSetVectorLimits()`
`ggEnqVectorLimits()`
	Set/enquire vector strength clipping limits

`ggSetVectorAttribs()`
`ggEnqVectorAttribs()`
	Set/enquire vector scaling factor

`ggRestoreVectorSettings()`	Restores Vector Chart default attributes
`ggAddVectors()`	Draws grid of vectors

Vector Chart Components

The following describes the low level routines provided by GINOGRAF for the drawing of Vector Charts. The following steps are required to set up the required mapping between vector data and drawing units as well as drawing the chart itself:

- Position, scale and optionally label and draw graphical axes
- Optionally define Vector Chart scaling and/or mapping
- Draw grid of vectors

Vector Chart Axes Definition and Display

Vector Charts are drawn with reference to the currently defined graphical axes system in that the grid of vectors is mapped onto either the complete area defined by the intersection of the horizontal (X) and vertical (Y) axes physical limits or a sub-area defined in graphical coordinates. The scaling of axes may also be required to define vector position information for each chart.

The default area and scaling of the graphical axes system may be altered using the axis definition routines ggSetAxesPos() and ggSetAxesScaling(). Equally the actual display and labelling of these axes is also under the control of the user.

Full description of axes definition, display and titling is found elsewhere (see page 19).

Vector Chart Attributes

Vector Chart Mapping

By default the Vector Chart occupies the area defined by the intersection of the horizontal (X) and vertical (Y) axes physical limits. The routine ggSetVectorChartFrame() can be used to define an alternative area onto which the Vector Chart will be mapped:

```
ggSetVectorChartFrame(limits)
```

where **limits** defines the required area in graphical coordinates. The graphical coordinate system is that set up by the most recent calls to ggSetAxesScaling().

Vector Clipping and Scaling

By default all vectors with absolute strength greater than zero are drawn and vectors are scaled so that the maximum strength is represented by a vector which occupies one unit square on the Vector Chart (that is the largest length possible without vectors overlapping). Two routines are provided to change these defaults:

```
ggSetVectorLimits(smin,smax)
```

ggSetVectorLimits() defines an upper and lower limit of absolute vector strength, outside which vectors are not drawn.

```
ggSetVectorAttribs(pos,vecmin,vecmax,factor)
```

The routine ggSetVectorAttribs() sets an additional scaling factor by specifying the lower (**vecmin**) and upper (**vecmax**) absolute vector strengths which will be represented by a zero length vector (not drawn) and a unit length vector respectively. An additional overall vector scaling factor can also be applied through the argument **factor**. Thus, if vector strengths range from 0.0 to 10.0 the following settings have the described effect on the length of vectors:

	Limit below which vectors are not drawn	**Strength represented by unit length vector**	**Length of vector for strength 10.0**
Default settings	0.0	10.0	1 * unit length
ggSetVectorAttribs(GCENTRE,2.0,5.0,1.0)	2.0	5.0	8/3 * unit length
ggSetVectorAttribs(GCENTRE,2.0,5.0,2.0)	2.0	2.5	16/3 * unit length

The argument **pos** determines whether the vector tail (**pos**=GTAIL), centre (**pos**=GCENTRE) or head (**pos**=GHEAD) is positioned at the grid intersection.

Resetting Attributes

The routine ggRestoreVectorSettings() resets the mapping, scaling and clipping Vector Chart attributes to their default settings:

ggRestoreVectorSettings()

Enquiring Attributes

Three enquiry routines are provided to enquire the current setting of the Vector Chart attributes. These routines match the above setting routines and can be used at any time within a GINOGRAF application. The routines are:

ggEnqVectorChartFrame(limits)

ggEnqVectorLimits(smin,smax)

ggEnqVectorAttribs(pos,vecmin,vecmax,factor)

Vector Chart Drawing

The routine to draw the Vector Chart is:

ggAddVectors(nx,ny,vecarray,head)

where **vecarray** is a two dimensional array of dimension **nx** by **ny** and type GVECTOR. The element **vecarray.direc** contains a grid of vector directions in degrees measured from the 3 o'clock (positive X axis) position; **vecarray.stren** contains a grid of vector strengths and **vecarray.col** contains a grid of colour index numbers. Negative strength values reverse the corresponding direction held in **vecarray.direc**.

The argument **head** determines whether the vector has an open (**head**=GOPEN), closed (**head**=GCLOSED) or filled (**head**=GSOLID) arrow head at the end of each vector.

The Vector Chart is mapped onto either the area defined by the latest call to ggSetVectorChartFrame() or the area defined by the intersection of the horizontal and vertical graphical axes if ggSetVectorChartFrame() has not been called. This area is divided into **nx** by **ny** grid points onto which each vector is drawn. The vector representing the element (1,1) is located at lower left corner of this area.

The following example shows a program using the routine ggAddVectors() (this example does not use the **vecarray.col** element to best effect because of the monochrome output).

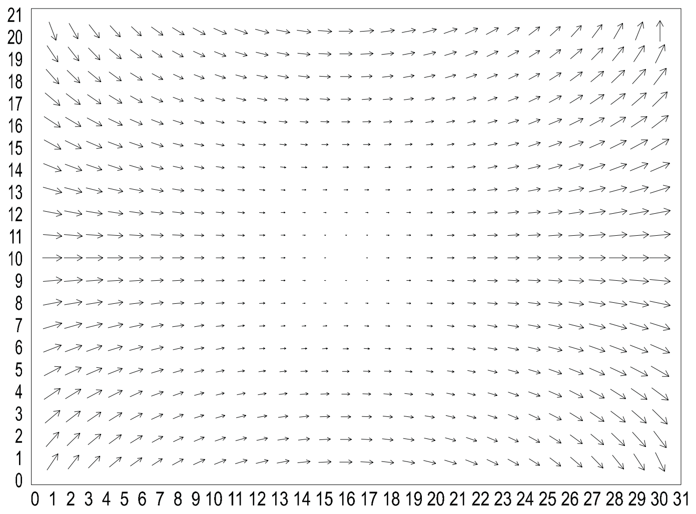
C Code

```
/*  VECTOR CHART EXAMPLE */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   int     i, j;
   GVECTOR vecs[30][20];
   GLIMIT  lims={1.0,30.0,1.0,20.0};

/* ENTER GINO & INITIALIZE DEVICE */
   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* SET AXES RANGES FROM 0 TO 31/21 */
   ggSetAxesScaling(GLINEARTYPE3,31,0.0,31.0,GXAXIS);
   ggSetAxesScaling(GLINEARTYPE3,21,0.0,21.0,GYAXIS);
   ggAddGrid(GNONE,GNONE,GANNOTATION,GANNOTAION);

   for (i=0; i<30; i++) {
     for (j=0; j<20; j++) {
       vecs[i][j].direc=asin((float)((j-10)*(i-15))/150.0)
         *57.296;
       vecs[i][j].stren=max(abs(i-15),abs(j-10))*10.0;
       vecs[i][j].col=1;
     }
   }

/* SET BOUNDARY */
   ggSetVectorChartFrame(&lims);
```

```
        /* DRAW VECTOR CHART */
           ggAddVectors(30,20,vecs,GOPEN);

           gSuspendDevice();
           gCloseGino();
           return(0);
        }
```

F90 Code

```
        !  VECTOR CHART EXAMPLE
        use gino_f90
        use graf_f90


          integer i, j
          type (GVECTOR), dimension(30,20) :: vecs
          type (GLIMIT) :: lims = GLIMIT(1.0,30.0,1.0,20.0)

        ! ENTER GINO & INITIALIZE DEVICE
          call gOpenGino
          call xxxxx
          call ggSetGraphCharMode(GGINOMODE)

        ! SET AXES RANGES FROM 0 TO 31/21
          call ggSetAxesScaling(GLINEARTYPE3,31,0.0,31.0,GXAXIS)
          call ggSetAxesScaling(GLINEARTYPE3,21,0.0,21.0,GYAXIS)
          call ggAddGrid(GNONE,GNONE,GANNOTATION,GANNOTAION)

          do i=1,30
            do j=1,20
              vecs(i,j)%direc=asin(real((j-10)*(i-15))/150.0)*57.296
              vecs(i,j)%stren=max(abs(i-15),abs(j-10))*10.0
              vecs(i,j)%col=1
            end do
          end do

        ! SET BOUNDARY
          call ggSetVectorChartFrame(lims)

        ! DRAW VECTOR CHART
          call ggAddVectors(30,20,vecs,GOPEN)

          call gSuspendDevice
          call gCloseGino
          stop
          end
```



Example Vector Chart Output

Chapter 6

POLAR CHARTS

Introduction to Polar Charts

The production of Polar Charts is outlined in this chapter. The GINOGRAF definition of a Polar Chart is a two dimensional discrete axis graph showing points at a distance R from the origin at an angle theta (from the 3 o'clock position - anti clockwise). The routines are split up into two sections, the complete drawing routine displays a Polar Chart with a single routine call; and component routines which build up Polar Charts in a modular fashion.

Summary of the Polar Chart facilities:

`ggPlotXYPolarChart()`	Draws complete Polar Chart and axes with automatically scaled data
`ggSetPolarChartAttribs()`	Sets position, radius and scaling of Polar Chart
`ggDrawPolarAxes()`	Draws individual polar axis

Complete Polar Chart

The routine described in this section is complete in itself. The user simply provides a set of data in an array of type GPOINT and makes a single routine call. The position and scaling of the Polar Chart is calculated automatically and output is drawn to fit the current graph drawing area.

The Complete Polar Chart routine is:

ggPlotXYPolarChart(npts,points,style)

The data coordinates, passed to the routine via the array **points** (of type GPOINT), must be converted from polar values to linear values within the graphical axes coordinates as follows :

```
points.x = radius  * COS (theta)

points.y  = radius  * SIN (theta)
```

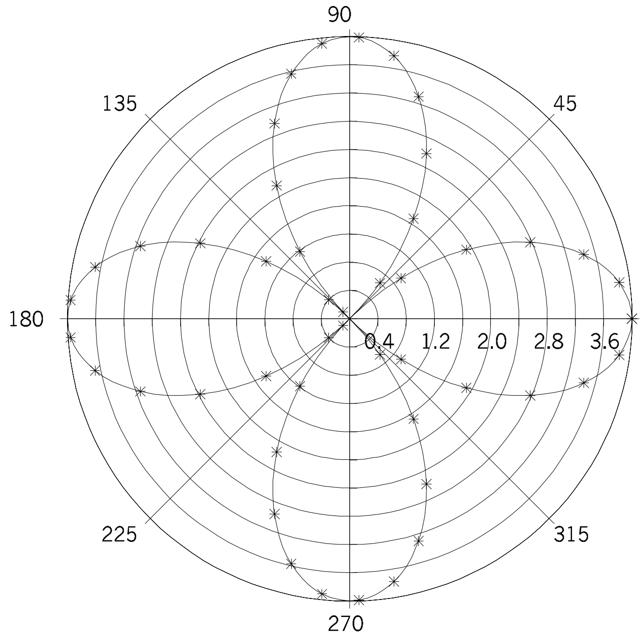
where radius and theta are the radius and angle in radians of the polar coordinates.

The data points may be represented in various ways depending on the value of **style** as follows :

style	**Lines**	**Marker**
-GAKIMA	Akima Curve	Asterisk
-GSPLINE	Spline Curve	Asterisk
-GCUBIC	Curve	Asterisk
-GSTRAIGHT	Straight Line	Asterisk
GSYMBOLS	None	Asterisk
GSTRAIGHT	Straight Line	None
GCUBIC	Curve	None
GSPLINE	Spline Curve	None
GAKIMA	Akima Curve	None

The Polar Chart is automatically scaled and annotated to fit centrally within the available drawing area, giving the full 360° radius and containing all the points in the data array. The numerical annotation on the radial axis is subject to the current settings of format and annotation control set by ggSetAxesAnnotation() and ggSetAxesAttribs().

An example of output produced by ggPlotXYPolarChart() is shown in the figure below.



Use of Complete Chart Routine ggPlotXYPolarChart

The following code shows the program that generated it followed by an equivalent program using the low-level component routines described in the next section. While axis and graph titles can be added after the routine ggPlotXYPolarChart() has been called, the full flexibility of layout and style can only be achieved using these component routines as the routine ggPlotXYPolarChart() is provided to present user data as quickly as possible with the minimum of effort.

C Code

```
/*  USE OF COMPLETE POLAR CHART ROUTINE -
   ggPlotXYPolarChart() */
#include <gino-c.h>
#include <graf-c.h>

/* DEFINE CONSTANTS */
#define N 48
#define PI 3.1415926
#define RAD 2.0*PI/(N-1)
```

```
         int main(void) {
            GPOINT pnts[N];
            float r, theta;
            int I;

         /* CALCULATE FUNCTION VALUES */
            for (i=0; i<N; i++) {
              theta=(i-1)*RAD;
              r=4*cos(2*theta);
              pnts[I].x=r*cos(theta);
              pnts[I].y=r*sin(theta);
            }

         /* DRAW POLAR CHART */
            gOpenGino();
            xxxxx();
            ggSetGraphCharMode(GGINOMODE);

            ggPlotXYPolarChart(N,pnts,-GCUBIC);

            gSuspendDevice();
            gCloseGino();
            return(0);
         }
```

```
         /*  USING LOW LEVEL COMPONENT ROUTINES */
         #include <gino-c.h>
         #include <graf-c.h>

         /*DEFINE CONSTANTS */
         #define N 48
         #define PI 3.1415926
         #define RAD 2.0*PI/(N-1)

         int main(void) {
            GPOINT pnts[N];
            float r, theta, polrad;
            int i, flg;
            GLIMIT lims;

         /* CALCULATE FUNCTION VALUES */
            for (i=0; i<N; i++) {
              theta=(i-1)*RAD;
              r=4*cos(2*theta);
              pnts.x[i]=r*cos(theta);
              pnts.y[i]=r*sin(theta);
            }

         /* DRAW POLAR CHART */
            gOpenGino();
            xxxxx();
            ggSetGraphCharMode(GGINOMODE);

         /* ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE */
            ggEnqPlotFrame(&flg,&lims);

            polrad=0.375*abs(min(lims.xmax-lims.xmin,lim.ymax-lims.ymin));

         /* SET UP AXIS POSITION AND SCALES */
            ggSetPolarChartAttribs((lims.xmin+lims.xmax)/2.0,
                (lims.ymin+lims.ymax)/2.0,polrad,GLINEARTYPE3);
```

```
/* DRAW AXES */
   ggDrawPolarAxes(GCARDINAL,GTICKSANDCIRCLES,GANNOTATION,
         10,4.0,GRAXIS);
   ggDrawPolarAxes(GINTERMEDIATE,GTICKSANDRADII,GANNOTATION,
          8,360.0,GTHETAAXIS);

/* DRAW CURVE THROUGH POINTS AND ADD SYMBOLS */
   ggAddGraphCurve(N,pnts);
   ggAddGraphMarkers(N,pnts,GSTAR,0);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 Code

```
!  USE OF COMPLETE POLAR CHART ROUTINE -
! ggPlotXYPolarChart()
use gino_f90
use graf_f90

! DEFINE CONSTANTS
  integer, parameter :: N=48
  real, parameter :: PI=3.1415926
  real, parameter :: RAD=2.0*PI/(N-1)

  type (GPOINT), dimension(N) :: pnts
  real r, theta
  integer I

! CALCULATE FUNCTION VALUES
  do i=1, N
    theta=(i-1)*RAD
    r=4*cos(2*theta)
    pnts(I)%x=r*cos(theta)
    pnts(I)%y=r*sin(theta)
  end do

! DRAW POLAR CHART
  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

  call ggPlotXYPolarChart(N,pnts,-GCUBIC)
  call gSuspendDevice
  call gCloseGino
  stop
  end
```

```
!  USING LOW LEVEL COMPONENT ROUTINES
use gino_f90
use graf_f90

!DEFINE CONSTANTS
  integer, parameter :: N=48
  real, parameter :: PI=3.1415926
  real, parameter ::   type (GPOINT), dimension(N) :: pnts
  real r, theta, polrad
  integer i, flg
  type (GLIMIT) lims

! CALCULATE FUNCTION VALUES
  RAD=2.0*PI/(N-1)
  do i=1, N
    theta=(i-1)*RAD;
    r=4*cos(2*theta);
    pnts(i)%x=r*cos(theta);
    pnts(i)%y=r*sin(theta);
  end do

! DRAW POLAR CHART
  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

! ENQUIRE GINOGRAF DRAWING LIMITS AND CHARACTER SIZE
  call ggEnqPlotFrame(flg,lims)

  polrad=0.375* abs(min(lims%xmax-lims%xmin, &
    lims%ymax-lims%ymin))

! SET UP AXIS POSITION AND SCALES
  call ggSetPolarChartAttribs((lims%xmin+lims%xmax)/2.0, &
    (lims%ymin+lims%ymax)/2.0,polrad,GLINEARTYPE3)

! DRAW AXES
  call ggDrawPolarAxes(GCARDINAL,GTICKSANDCIRCLES, &
    GANNOTATION,10,4.0,GRAXIS)
  call ggDrawPolarAxes(GINTERMEDIATE,GTICKSANDRADII, &
    GANNOTATION,8,360.0,GTHETAAXIS)

! DRAW CURVE THROUGH POINTS AND ADD SYMBOLS
  call ggAddGraphCurve(N,pnts)
  call ggAddGraphMarkers(N,pnts,GSTAR,0)

  call gSuspendDevice
  call gCloseGino
  stop
  end
```

Polar Chart Components

This section describes the component routines provided by GINOGRAF for drawing Polar Charts. The routines provide the user with the tools to build up more complex and sophisticated Polar Charts than those created using the complete drawing routine.

Each Polar Chart is built up in a modular fashion, making use of individual routines to:

- Position and scale axes
- Draw axes
- Represent polar data sets with symbols, lines, and curves

Polar Chart axes use a special form of graphical axes which consists of a pair of axes crossing at a centre point. The centre point represents the origin of the polar axis coordinate system (radius = 0.0).

Positioning and Scaling

The routine ggSetPolarChartAttribs() defines the centre, radius, and the type of scaling of polar axes:

```
ggSetPolarChartAttribs(xorp,yorp,radlen,scale)
```

The centre of the Polar Chart (**xorp**, **yorp**) which represents the origin of the axes, and the radius of the Polar Chart **radlen** are defined in user space coordinates.

The Polar Chart may be scaled in three ways, all linear, depending on the value of **scale**. The styles vary in the way that the number of intervals (**nints**) and the end range of values (**vendp**) are represented in the annotation when the axes are drawn using ggDrawPolarAxes().

The annotation for theta extends beyond the circumference of the largest circular axis. It is advisable, therefore, that there is enough room around the Polar Chart if theta annotation is to be displayed.

If the positioning and scaling of the axes is omitted the default position and scale takes effect (see page 16).

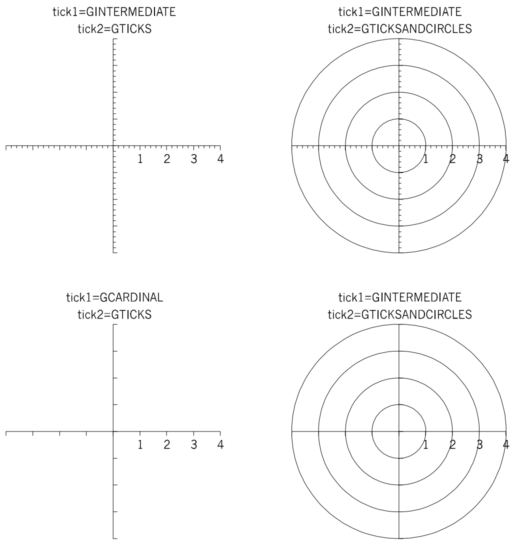
Polar Axes Drawing

The routine ggDrawPolarAxes() is used to draw either the radial or theta axes:

```
ggDrawPolarAxes(tick1,tick2,val,nintp,vendp,rorth)
```

where **rorth** determines which axes is to be drawn. **rorth**=GRAXIS draws the radial axis and **rorth**=GTHETAAXIS draws the theta axis. ggDrawPolarAxes() draws the requested axes at the position and size set by the last call to ggSetPolarChartAttribs(), or at the centre of the current drawing area if this has not been called.

vendp sets the numeric limit of either axis which is measured in current units for the radial axis and degrees for the theta axis. The minimum limit is zero for both axes. **nintp** sets the number of intervals on either axis. How closely the routine ggDrawPolarAxes() adheres to the setting of **nintp** and **vendp**, is determined by the scale type set by the most recent call to ggSetPolarChartAttribs(). Radial annotation is shown along the axis representing zero degrees in the theta direction, and angular annotation is shown anticlockwise starting from zero degrees in the three o'clock position.



Polar Axes

tick1 and **val** determine whether tick marks and annotation are displayed on either axis. **tick2** controls the display of radial lines and circles at major tick mark intervals.

Polar Axes Annotation Control

As Polar Chart axes are a special form of graphical axes (see page 19), the same annotation control routines described in that section also affect Polar Chart axes.

Therefore ggSetAxesAnnotation() can be used to set the numeric format and scale type display and ggSetAxesAttribs() can be used to control the position, angle and justification of Polar Chart annotation.

The axes titling and labelling routines ggDrawAxesTitle() and ggDrawAxesLabels() can also be used with Polar Chart axes.

Polar Axes Enquiry

Both the Complete Drawing routine ggPlotXYPolarChart() and the component routine ggDrawPolarAxes() redefine the current settings for the position and scaling of graphical axes. The routines ggEnqAxesPos() and ggEnqAxesScaling() will therefore return the current axes settings for either type of graph.

If Polar Charts are to be used in conjunction with normal graphs, it should be realised that the Polar Chart drawing routines will redefine the position and scaling of graphical axes as set up with the routines ggSetAxesPos() and ggSetAxesScaling(). If particular settings need to be saved and restored, the axis enquiry routines ggEnqAxesPos() and ggEnqAxesScaling() can be used for this purpose.

Polar Chart Default Restoration

The routine used to restore the default polar axis definition is:

```
ggRestoreAxesSettings()
```

ggRestoreAxesSettings() has no arguments. It restores to their default values the polar axis positioning and scaling set using ggSetPolarChartAttribs().

Polar Chart Drawing

The data sets are drawn on the most recently defined polar axes, therefore many data sets may be displayed on a set of axes that have been defined once. Consequently, data sets may be added to a Polar Chart set up using the complete Polar Chart routine ggPlotXYPolarChart()

Data sets are represented on polar axes using the component graph drawing routines ggAddGraphPolyline(), ggAddGraphCurve(), ggAddGraphSpline(), ggAddGraphMarkers() or ggFillBelowDataset().

The data coordinates, passed to the routine via the array **points.x** and **points.y** of **npts** points, must be converted from polar values to linear values within the graphical axes coordinates as follows:

```
points.x = radius  * COS (theta)

points.y = radius  * SIN (theta)
```

where radius and theta are the radius and angle in radians of each polar coordinate.

An example of the way in which the values are converted is shown in the previous coding example.

Chapter 7

PIE CHARTS

Introduction to Pie Charts

The production of Pie Charts is covered in this chapter. GINOGRAF can display two-dimensional filled Pie Charts with centred or exploded segments containing text and numerical annotation.

When using the word "segment", GINOGRAF refers to an individual Pie Chart slice.

Summary of the Pie Chart facilities:

`ggSetPieChartFrame()`	Defines non-default Pie Chart position
`ggPlotPieChart()`	Draws annotated, filled Pie Chart
`ggAddPieChartSegment()`	Draws annotated, filled Pie Chart segment
`ggSetPieChartAnnotation()`	Sets Pie Chart annotation type
`ggSetPieChartBoxType()`	Sets internal annotation box type
`ggSetPieChartStartAngle()`	Defines start angle for Pie Chart display
`ggSetPieChartBoundSwitch()`	Sets Pie Chart boundary switch
`ggSetPieChartExplosion()`	Defines explosion factors for Pie Chart segments
`ggEnqPieChartSettings()`	Returns Pie Chart position and start angle
`ggEnqPieChartAnnotation()`	Returns Pie Chart annotation type
`ggRestorePieChartSettings()`	Restores default settings for Pie Chart display

Pie Chart Facilities

Two routines are provided to draw either complete Pie Charts or individual segments. The routines are tabled below showing the various facilities of each:

Routine	Boundary	Filling	Annotation	Output
`ggPlotPieChart()`	yes	yes	radial, internal, external	text, percentage, value
`ggAddPieChartSegment()`	yes	yes	radial, internal, external	text, percentage, value

All the Pie Chart routines are affected by the Pie Chart control routines, the defaults of which are found in Appendix A.

Three forms of Pie Chart annotation are provided:

- Radial annotation (type GRADIAL) is printed along the bisecting angle of a segment in such a way that it is readable from left to right when viewed from the 6 o'clock position (Y-axis negative direction). The start position is adjusted to ensure the first character fits between the edges of the segment. For small segments the character size may be reduced to half the current size in order to fit within the segment. However, for very small segments or long text strings the annotation may extend or even start outside the segment boundary.
- Internal annotation (type GINTERNAL) is printed horizontally within the segment. Combinations of text string, percentage, and/or value are printed above one another. If there is enough room the strings are positioned centrally within the segment, otherwise adjustments are made so that they still fit inside the segment. However, if the required strings cannot be satisfactorily fitted by this means then they are automatically printed outside the segment (as for external annotation).
- External annotation (type GEXTERNAL) prints the required output as a single string consisting of the required data items horizontally outside each segment with a connecting line.

The algorithms for the above annotation types have been created to work on any data, though, because each segment is annotated without any knowledge of other segments, sometimes external annotation may overlap. This may be avoided by altering one of the following :

- Character size
- Type of annotation (radial annotation can never overlap)
- Minimum percentage tolerance of annotated segments (See ggSetPieChartAnnotation())
- Length of strings being output
- Output format control of values (see - ggSetAxesAnnotation())
- Length of values prefixes and suffixes (see - ggSetValueTags())
- Ordering of data sets

A maximum of 50 segments is permitted in all the Pie Chart routines.

Pie Chart Size and Position

The default size (ie, radius) and position of Pie Charts and segments are determined by the current GINOGRAF drawing area. They are not affected by the axis definition routines ggSetAxesPos() and ggSetAxesScaling().

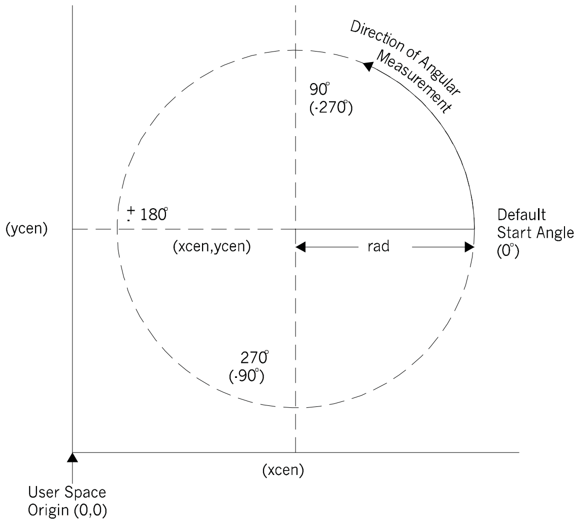
ggSetPieChartFrame() can be used to define an alternative Pie Chart size and position as follows:

```
ggSetPieChartFrame(rad,xcen,ycen)
```

where **rad** is the radius of the Pie Chart or segment in current units and **xcen**,**ycen** is the coordinate position of the centre of the Pie Chart in user space coordinates. Calls to ggSetPieChartFrame() provide control for calls to all the Pie Chart and segment routines. Successive calls to ggSetPieChartFrame() override previous Pie Chart definitions.

The current Pie Chart size and position, whether set by default or by ggSetPieChartFrame() can be enquired through the routine ggEnqPieChartSettings().

If ggSetPieChartFrame() has been used, the default sizing and positioning can be restored using the routine ggRestorePieChartSettings().



Relationship of xcen, ycen & rad to drawing area

The relationship of **xcen**, **ycen** and **rad** to the drawing area / window is shown above.

Pie Chart Drawing

The Pie Chart routine for drawing, annotating and filling a Pie Chart is:

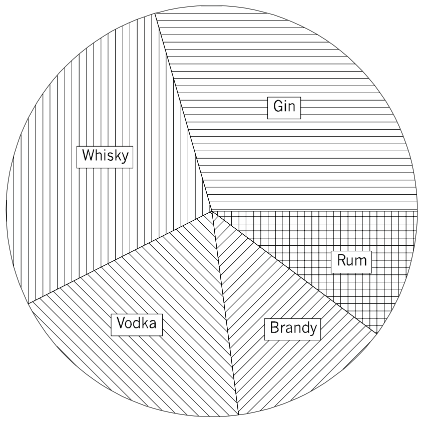
ggPlotPieChart(nsegs,value,string,fill,line)

The Pie Chart consists of **nsegs** segments with the data, annotation and fill styles taken from the arrays **value**, **string**, **fill**, and **line** each of which should contain **nsegs** elements. The segments sizes are automatically calculated by assigning each value in the array **value** a proportion of the complete circle in relation to its proportion of the sum of all the **nsegs** values in the array **value**, ie:

segment angle / 360° = value / total of values

Each segment is filled with the corresponding fill style and line style index contained in the **fill** and **line** arrays. Solid fill and various hatches and cross hatches are available. The default styles are shown in Appendix A, but they can be changed using the GINO line attribute routines. If an element of **fill** contains a number less than -1 (GHOLLOW), the corresponding Pie Chart segment is not filled.

The default annotation type is GINTERNAL - printed horizontally within the segment.



High Level Complete PieChart

C Code

```
/*  HIGH LEVEL PIECHART */
#include <gino-c.h>
#include <graf-c.h>

#define nsegs 5
int main(void) {
  float value[nsegs] = {81.0,77.0,54.0,35.0,28.0};

  int   fill[nsegs] = {1,2,3,4,5};
  int   line[nsegs] = {1,1,1,1,1};
  char  string[nsegs] = {“gin”,"whisky","vodka",
    “brandy”,"rum"};

  gOpenGino();
  xxxxx();
  ggSetGraphCharMode(GGINOMODE);
  ggPlotPieChart(nsegs,&value,&string,&fill,&line);
  gSuspendDevice();
  return(0);
}
```

F90 Code

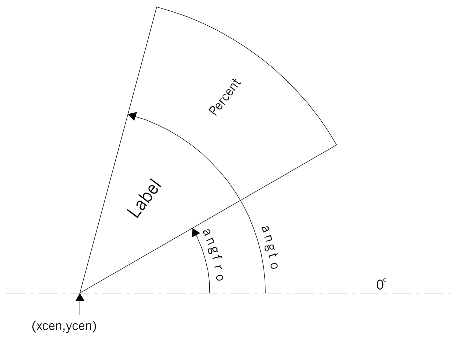
```
!  HIGH LEVEL PIECHART
use gino_f90
use graf_f90

  integer, parameter :: nsegs = 5

  real, dimension(nsegs) :: value = &
    (/81.0,77.0,54.0,35.0,28.0/)
  integer, dimension(nsegs) :: fill = (/1,2,3,4,5/)
  integer, dimension(nsegs) :: line = (/1,1,1,1,1/)
  character (len=6), dimension(nsegs) :: string = &
    (/'gin','whisky','vodka','brandy','rum'/)

  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)
  call ggPlotPieChart(nsegs,value,string,fill,line)
  call gSuspendDevice
  stop
  end
```

Single Segment Output

Where a single segment is required rather than a complete Pie Chart, the routine ggAddPieChartSegment() provides the same facilities as ggPlotPieChart():

ggAddPieChartSegment(angfro,angto,value,string,fill,line)

where **angfro** and **angto** are the start and finish angles (measured anticlockwise from the three o'clock position) of the segment. **value** and **string** are the single data value and annotation string corresponding to the segment. While there may not be any data relevant to the single segment, a dummy value must be supplied. **fill** and **line** are the fill and line style indices for the segment as with ggPlotPieChart(). The segment is centred at the current Pie Chart centre.



Pie Chart Segment

Pie Chart Annotation Control

Control of the annotation for Pie Charts and segments is made with the routine:

ggSetPieChartAnnotation(type,txt,per,val,tol) where **type** sets the annotation type. The choices are GRADIAL, GINTERNAL (default) and GEXTERNAL. The three arguments **txt**, **per** and **val** determine which of the three annotation data items are printed with each segment, the text string (in **string**), the calculated percentage value of the segment and the data value itself (in **value**). The data item is included if the corresponding flag is equal to GTEXT, GPERCENT and GDATA respectively, and omitted if equal to GNOTEXT, GNOPERCENT and GNODATA. **tol** is a tolerance value (in percentage terms) below which segments are not annotated at all. This is useful to control annotation of very small segments where data is insignificant.

The following example shows radial and external annotation types with just the percentage value printed:

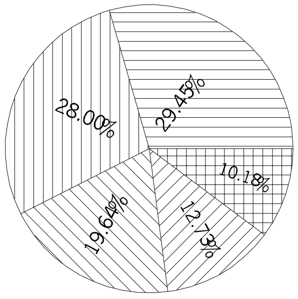
C Code

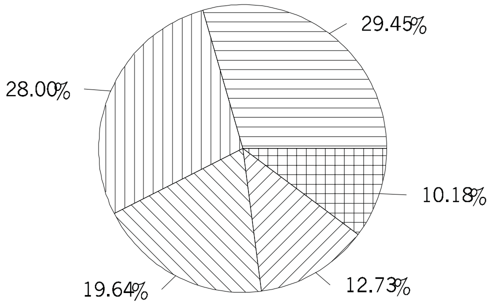
```
/*  RADIAL AND EXTERNAL ANNOTATION TYPES */
#include <gino-c.h>
#include <graf-c.h>

#define nsegs=5

int main(void) {
  int papty
  GDIM paper
  GLIMIT lims
  float value[nsegs] = {81.0,77.0,54.0,35.0,28.0};

  int fill[nsegs] = {1,2,3,4,5};
  int line[nsegs] = {1,1,1,1,1};
  char string[nsegs] = {“gin”,"whisky","vodka",
    “brandy”,"rum"};

  gOpenGino();
  xxxxx();
  gEnqDrawingLimits(&paper,&papty);
  gSetWindowMode(GON2D);
  ggSetGraphCharMode(GGINOMODE);

  lims.xmin=0.0;
  lims.xmax=0.4*paper.xpap;
  lims.ymin=0.0;
  lims.ymax=paper.ypap;
  ggSetPlotFrame(&lims);
  ggSetPieChartAnnotation(GRADIAL,GNOTEXT,
    GPERCENT,GNODATA,0.0);
  ggPlotPieChart(nsegs,&value,&string,&fill,&line);

  lims.xmin=0.5*paper.xpap;
  lims.xmax=0.9*paper.xpap;
  lims.ymin=0.0;
  lims.ymax=paper.ypap;
  ggSetPlotFrame(&lims);
  ggSetPieChartAnnotation(GEXTERNAL,GNOTEXT,
    GPERCENT,GNODATA,0.0);
  ggPlotPieChart(nsegs,&value,&string,&fill,&line);
```

```
        gSuspendDevice();
        return(0);
      }
```

F90 Code

```
      !  RADIAL AND EXTERNAL ANNOTATION TYPES
      use gino_f90
      use graf_f90

        integer, parameter :: nsegs=5

        integer papty
        type (GDIM) paper
        type (GLIMIT) lims
        real, dimension(nsegs) :: value = &
          (/81.0,77.0,54.0,35.0,28.0/)
        integer, dimension(nsegs) :: fill = (/1,2,3,4,5/)
        integer, dimension(nsegs) :: line = (/1,1,1,1,1/)
        character (len=6), dimension(nsegs) :: string = &
          (/'gin','whisky','vodka','brandy','rum'/)

        call gOpenGino
        call xxxxx
        call gEnqDrawingLimits(paper,papty)
        call gSetWindowMode(GON2D)
        call ggSetGraphCharMode(GGINOMODE)

        lims%xmin=0.0
        lims%xmax=0.4*paper%xpap
        lims%ymin=0.0
        lims%ymax=paper%ypap
        call ggSetPlotFrame(lims)
        call ggSetPieChartAnnotation(GRADIAL,GNOTEXT, &
          GPERCENT,GNODATA,0.0)
        call ggPlotPieChart(nsegs,value,string,fill,line)

        lims%xmin=0.5*paper%xpap
        lims%xmax=0.9*paper%xpap
        lims%ymin=0.0
        lims%ymax=paper%ypap
        call ggSetPlotFrame(lims)
        call ggSetPieChartAnnotation(GEXTERNAL,GNOTEXT, &
          GPERCENT,GNODATA,0.0)
        call ggPlotPieChart(nsegs,value,string,fill,line)

        call gSuspendDevice
        stop
        end
```





Example of Radial and External annotation

The format of the data values printed as part of Pie Chart annotation is controlled with the axis annotation control routine ggSetAxesAnnotation() (N.B. the value of **xory** must be set to GYAXIS to control Pie Chart annotation):

ggSetAxesAnnotation(ndp,npower,axty,GYAXIS)

By default, values are output with up to two decimal places, but **ndp** may be set to adjust this if required. The use of ggSetAxesAnnotation() for Pie Charts is to enable values output in the chart to match values displayed on the Y axis of a graph or chart if required. But it should be noted that **axty** has no effect on Pie Charts and if **npower** is non zero no scale factor is output with the Pie Chart even though the values are multiplied by the appropriate power of 10. Again this can be used to match axes output if required.

All the data values output by ggPlotPieChart() may by augmented by a common prefix and/or suffix string of up to 30 characters as defined by the utility routine:

ggSetValueTags(prefix,suffix)

By default, both strings are null and if either string needs to be set to null, the string `*.' should be used as shown in the example below. Percentage values are always followed by a `%' sign and cannot have a prefix or suffix string appended to them.

The example below shows the use of format control and a common suffix string appended to data values output within a Pie Chart.

C Code

```
/*  ANNOTATION CONTROL WITHIN PIECHART */
#include <gino-c.h>
#include <graf-c.h>

#define nsegs 5
int main(void) {
   float value[nsegs] = {81.0,77.0,54.0,35.0,28.0};

   int fill[nsegs] = {1,2,3,4,5};
   int line[nsegs] = {1,1,1,1,1};
   char string[nsegs = {”gin”,”whisky”,”vodka”,”brandy”,”rum”};

   gOpenGino();
   xxxxx();
   ggSetGraphCharMode(GGINOMODE);

/* SET ANNOTATION TYPE GINTERNAL WITH STRING AND DATA VALUE OUTPUT
*/
   ggSetPieChartAnnotation(GINTERNAL,GTEXT,GNOPERCENT,GDATA,0.0);

/* FORCE ONE DECIMAL PLACE ON VALUE OUTPUT */
   ggSetAxesAnnotation(-1,0,GNOSCALE,GYAXIS);

/* ADD SUFFIX STRING TO VALUE OUTPUT */
   ggSetValueTags(“*.”,"gal");

/* OUTPUT PIECHART */
   ggPlotPieChart(nsegs,&value,&string,&fill,&line);

   gSuspendDevice();
   return(0);
}
```

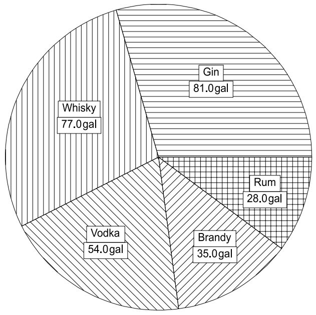
F90 Code

```
!  ANNOTATION CONTROL WITHIN PIECHART
use gino_f90
use graf_f90

  integer, parameter :: nsegs = 5

  real, dimension(nsegs) :: value = &
    (/81.0,77.0,54.0,35.0,28.0/)
  integer, dimension(nsegs) :: fill = (/1,2,3,4,5/)
  integer, dimension(nsegs) :: line = (/1,1,1,1,1/)
  character (len=6), dimension(nsegs) :: string = &
    (/’gin’,’whisky’,’vodka’,’brandy’,’rum’/)

  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

! SET ANNOTATION TYPE GINTERNAL WITH STRING AND
! DATA VALUE OUTPUT
  call ggSetPieChartAnnotation(GINTERNAL,GTEXT, &
    GNOPERCENT,GDATA,0.0)

! FORCE ONE DECIMAL PLACE ON VALUE OUTPUT
  call ggSetAxesAnnotation(-1,0,GNOSCALE,GYAXIS)
```

```
        ! ADD SUFFIX STRING TO VALUE OUTPUT
          call ggSetValueTags('*.','gal')

        ! OUTPUT PIECHART
          call ggPlotPieChart(nsegs,value,string,fill,line)

          call gSuspendDevice
          stop
          end
```

Pie Chart Annotation Box

By default, the internal Pie Chart annotation (type GINTERNAL) is outlined by a series of boxes around each data item and masked against any segment filling. These options can be changed with the routine:



Annotation control within piechart

ggSetPieChartBoxType(type,fill,line)

The following settings of **type** are permitted:

```
= GNONE                No filling/masking or boxes drawn
= GBOXED               No filling/masking but boxes are drawn
= GFILLED              Filling/masking done but no boxes draw
= GFILLED & GBOXED     Filling/masking done and boxes drawn
                       (default)
```

Where **type** = GNONE or BOXED, the box areas are not filled/masked and the annotation is drawn over any segment filling. Where **type** = GFILLED or (GFILLED & GBOXED), the box areas are filled with the specified fill and line styles, **fill** and **line**, and the annotation is drawn over this. The box area may be left unfilled (ie, masked) by setting **fill** to a negative number.

Pie Chart Control

The following routines control the position and form of all the Pie Chart routines in this Section:

- Start Angle Definition
- Pie Chart Boundary
- Pie Chart Explosion
- Pie Chart Enquiry
- Restoration of Defaults

Start Angle Definition

The routine ggSetPieChartStartAngle() defines the start angle of the first segment when drawing a Pie Chart using the routine ggPlotPieChart().

```
ggSetPieChartStartAngle(angle)
```

ggSetPieChartStartAngle() sets the start angle of the Pie Chart measured in degrees anticlockwise from the three o'clock (X-axis positive) position. The default angle is 0.0.

Pie Chart Boundary

The routine ggSetPieChartBoundSwitch() can be used to switch off the drawing of segment boundaries if required:

```
ggSetPieChartBoundSwitch(switch)
```

switch is a flag to determine the state of the Pie Chart boundary drawing. If **switch** = GOFF, future segments drawn by ggPlotPieChart() or ggAddPieChartSegment() will not include boundaries. Setting **switch** = GON restores the drawing of boundaries.

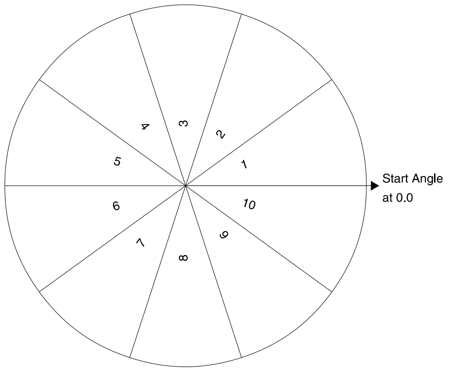
Pie Chart Explosion

Pie chart explosion can be used on Pie Charts using the routine ggPlotPieChart(). The routine ggSetPieChartExplosion() extracts some or all of the segments of a whole Pie Chart as follows:

```
ggSetPieChartExplosion(num,list,factor)
```

list is an array of dimension **num** containing the drawing sequence numbers of the segments to be extracted (ie, if **list** contains the numbers 1, 3 and 4, the first, third and fourth segments going anticlockwise from the start angle will be extracted).

factor is an array of dimension **num** giving the explosion factor for each of the segments identified in **list**. Each segment is extracted from the centre of the Pie Chart by a distance **factor** * radius, where radius is the current radius of the Pie Chart (ie, if **list** contains 1,3 and 4, and **factor** contains 0.1, 0.1, 0.25, the first and third segments would be extracted by one tenth of their radius, and the fourth segment would be extracted by one quarter of its radius as shown in the example below).



Pie Chart Segment numbering

C Code

```
        /*  PIE CHART EXPLOSION */
        #include <gino-c.h>
        #include <graf-c.h>

        #define nsegs=5
        int main(void) {
           float value[nsegs] = {81.0,77.0,54.0,35.0,28.0};
           int fill[nsegs] = {1,2,3,4,5};
           int line = {1,1,1,1,1};
           char string[nsegs] = {”gin”,”whisky”,”vodka”,”brandy”,”rum”};
           int list[3] = {1,3,4};
           float factor[3] = {0.1,0.1,0.25};

           gOpenGino();
           xxxxx();
           ggSetGraphCharMode(GGINOMODE);

        /* SET EXPLOSION FACTORS */
           ggSetPieChartExplosion(3,&list,&factor);

        /* DRAW PIECHART */
           ggPlotPieChart(nsegs,&value,&string,&fill,&line);

           gSuspendDevice();
           gCloseGino();
           return(0);
        }
```

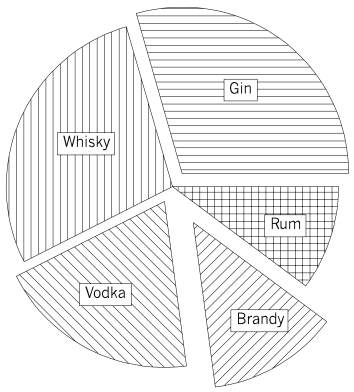
F Code

```
!  PIE CHART EXPLOSION
use gino_f90
use graf_f90

  integer, parameter :: nsegs=5
  real, dimension(nsegs) :: value = &
    (/81.0,77.0,54.0,35.0,28.0/)
  integer, dimension(nsegs) :: fill = (/1,2,3,4,5/)
  integer, dimension(nsegs) :: line = (/1,1,1,1,1/)
  character (len=6), dimension(nsegs) :: string = &
    (/'gin','whisky','vodka','brandy','rum'/)
  integer, dimension(3) :: list = (/1,3,4/)
  real, dimension(3) :: factor = (/0.1,0.1,0.25/)

  call gOpenGino
  call xxxxx
  call ggSetGraphCharMode(GGINOMODE)

! SET EXPLOSION FACTORS
  call ggSetPieChartExplosion(3,list,factor)

! DRAW PIECHART
  call ggPlotPieChart(nsegs,value,string,fill,line)

  call gSuspendDevice
  call gCloseGino
  stop
  end
```

If the radius is defined by default, sufficient space will be allowed for the extracted segments. However, if ggSetPieChartFrame() is called, the user should ensure that the radius is sufficiently small to allow for the explosion factors.

Calls to ggSetPieChartExplosion() require the user to establish a separate list of exploded segments. One way of avoiding this is to always define the size of **list** and **factor** as the number of segments in the Pie Chart, and assign **factor** = 0.0 for any segment which is not exploded.

Once called, ggSetPieChartExplosion() remains active until called again with **num** = 0. This resets the explosion list so that further calls to ggPlotPieChart() produce no exploded segments.



Example of Pie Chart explosion

Pie Chart Enquiry

Two enquiry routines are provided to return the current Pie Chart settings, ggEnqPieChartSettings() and ggEnqPieChartAnnotation():

ggEnqPieChartSettings(<u>radius</u>,<u>origin</u>,<u>angle</u>)

where **radius** gives the current radius of the Pie Chart in current drawing units and **origin.x**, **origin.y** give the X and Y coordinates of the Pie Chart centre expressed in user space coordinates **angle** gives the current start angle of the Pie Chart in degrees. These arguments are set by the routines ggSetPieChartFrame() and ggSetPieChartStartAngle().

ggEnqPieChartAnnotation(<u>type</u>,<u>txt</u>,<u>per</u>,<u>val</u>,<u>tol</u>)

ggEnqPieChartAnnotation() returns the current Pie Chart annotation settings as set by ggSetPieChartAnnotation().

Pie Chart Default Restoration

The routine to restore all Pie Chart settings to their defaults is:

ggRestorePieChartSettings()

which has no arguments. The routine resets the default Pie Chart position and size setting to that based on the current GINO window limits. It also restores the default Pie Chart annotation settings, Pie Chart annotation box, Pie Chart boundary, Pie Chart start angle and explosion factors to their respective defaults.

Chapter 8

TEXT CHARTS

Introduction to Text Charts

The routines described in this chapter provide facilities to present text strings, various forms of data and graphical items in related columns. They can be used to annotate other graph forms, for example, adding a key or legend, or to present data in tabular form in their own right. There are seven forms of Text Charts.

Summary of the Text Chart facilities:

`ggSetTextChartAttribs()` `ggEnqTextChartAttribs()`	Sets/enquires Text Chart size, annotation and header attributes
`ggDisplayStringColumn()`	Displays column of strings
`ggDisplayValueColumn()`	Displays column of values
`ggDisplayGeneratedColumn()`	Displays column of generated values
`ggDisplayPercentageColumn()`	Displays column of percentage values
`ggDisplayMarkerColumn()`	Displays column of symbols
`ggDisplayLineColumn()`	Displays column of boxes with line styles
`ggDisplayFillColumn()`	Displays column of boxes with fill styles

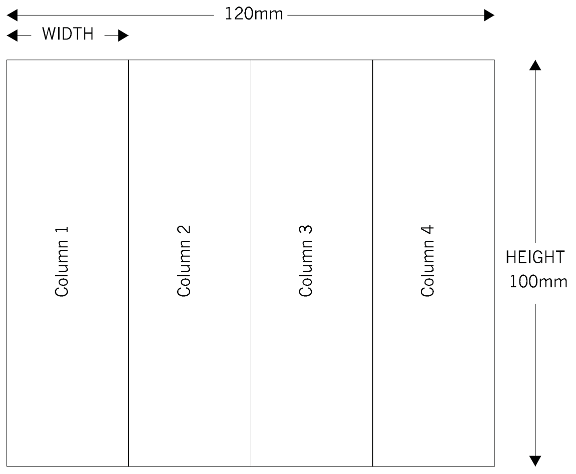
Text Chart Layout

When planning the layout of a Text Chart it is necessary to know the space available and decide a number of general points concerned with the form of the Text Chart. For example, if an area of 120mm x100mm is available and 4 columns are required, the dimensions of each column can easily be calculated. Each column is divided into a number of cells equal to the number of items to be output plus one extra cell if a header box is also required. Other attributes of the chart include the string and/or value justification and line style of the frame box itself. The routine ggSetTextChartAttribs() is used to set these attributes and ggEnqTextChartAttribs() returns the current settings.

```
ggSetTextChartAttribs(width,height,jushor,head,line)

ggEnqTextChartAttribs(width,height,jushor,head,line)
```

where **width** and **height** are the column dimensions and **jushor** is the string and/or value justification within the cell. **head** controls whether a header box is required within the column, the header string itself being provided in the output routine and is always centrally justified within the header box. **line** is the line style of the frame box outline (which is optional).

If the same attributes are required for a multi-column Text Chart, ggSetTextChartAttribs() would only be called once before each of the output routines, but if a different column size or justification is required for a new column ggSetTextChartAttribs() must be called before outputting the column. It would be desirable, but not obligatory, to keep the **height** and the header (**head**) the same.



Text Chart Layout

All the Text Chart routines use the current GINO character attributes for their text output, for both the header and the chart itself . The character size is **not** adjusted to fit the cell size and so it is the users' responsibility to set an appropriate size along with the setting of the chart attributes as this is obviously related to the cell size. The user should allow space for at least one extra character in the width of string, value or symbol charts and whatever space is required above and below in the height. If an insufficient cell size is set or the character size is too large, the string or value may overlap an adjacent cell or column depending on the justification setting. In the case of fill style and line style charts, these are always drawn to fit the current cell size with an appropriate spacing.

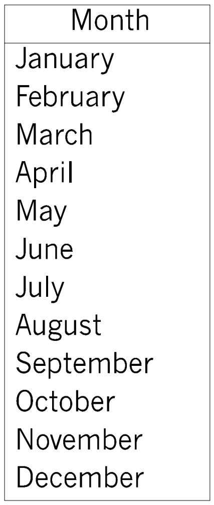
Even though the column size is set with the above routine, each output routine requires a start position allowing each column to be positioned anywhere in the drawing area, irrespective of the column size. Therefore if columns are required to be adjacent, the column width will also be required for updating the horizontal position for each column in turn. All sizes and positions in the Text Chart routines are in user space coordinates, **not** current graph coordinates.

String Text Chart

The routine ggDisplayStringColumn() is used to output a column of text strings:

```
ggDisplayStringColumn(x,y,nstr,string,header)
```

where **x**,**y** is the position of the top left corner of the column and **string** is a character array containing **nstr** strings. **header** is an optional header string which is output if headers are switched on with ggSetTextChartAttribs(). The following example shows how a single column Text Chart can be generated:



Month
January
February
March
April
May
June
July
August
September
October
November
December

Single Column Text Chart

C Code

```
/*  SINGLE COLUMN TEXT CHART */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   int papty, nrow;
   float cw, ch, rcolh, colh, x, y;
   GDIM paper;
   char *mons[12] = {”January”,”February”,”March”,”April”,
     ”May”,”June”,”July”,”August”,”September”,”October”,
     ”November”,”December”};

   gOpenGino();
   xxxxx();
   gEnqDrawingLimits(&paper,&papty);
   ggSetGraphCharMode(GGINOMODE);

/* SET UP TEXT CHART ATTRIBUTES */
   nrow = 12;
   cw = 5.0;
   ch = 5.0;
   gSetCharSize(cw,ch);

/* ENSURE COLUMN HEIGHT FITS IN DRAWING AREA */
   rcolh = (float)(nrow+1)*2.0*ch;
   colh = min(0.8*paper.ypap,rcolh);
   ggSetTextChartAttribs(10.0*cw,colh,GLEFT,GHEAD,1);

* DRAW CHART */
   x = 0.1*paper.xpap;
   y = 0.9*paper.ypap;
   ggDisplayStringColumn(x,y,nrow,mons,”Month”);

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 Code

```
!  SINGLE COLUMN TEXT CHART
use gino_f90
use graf_f90

  integer papty
  type (GDIM) paper
  character (len=9), dimension(12) :: mons = &
    (/’January’,’February’,’March’,’April’,’May’,’June’, &
      ’July’,’August’,’September’,’October’,’November’, &
      ’December’/)

  call gOpenGino
  call xxxxx
  call gEnqDrawingLimits(paper,papty)
  call ggSetGraphCharMode(GGINOMODE)
```

```
! SET UP TEXT CHART ATTRIBUTES
  nrow = 12
  cw = 5.0
  ch = 5.0
  call gSetCharSize(cw,ch)

! ENSURE COLUMN HEIGHT FITS IN DRAWING AREA
  rcolh = real(nrow+1)*2.0*ch
  colh = min(0.8*paper%ypap,rcolh)
  call ggSetTextChartAttribs(10.0*cw,colh,GLEFT, &
    GHEAD,1)

! DRAW CHART
  x = 0.1*paper%xpap
  y = 0.9*paper%ypap
  ggDisplayStringColumn(x,y,nrow,mons,'Month')

  call gSuspendDevice
  call gCloseGino
  stop
  end
```

Value Text Charts

Three routines are provided to output a column of values:

ggDisplayValueColumn(x,y,nval,values,header)

ggDisplayGeneratedColumn(x,y,nval,vbeg,vend,header)

ggDisplayPercentageColumn(x,y,nval,values,header)

ggDisplayValueColumn() is used to output a supplied set of real numbers held in the array **values**, ggDisplayGeneratedColumn() will output a set of **nval** generated values between **vbeg** and **vend** and ggDisplayPercentageColumn() will output the values held in the array **values** but re-calculated as a percentage of the total value of all the array locations. In each case the position **x,y** is the top left corner of the column and an optional header cell and the string **header**, is added if headers are switched on with ggSetTextChartAttribs().

These routines are provided to match the possible forms of numeric data that have been used with other graph forms, whether they be actual data, a data range or data supplied to a pie chart routine.

All the values are output using the current numerical format for the Y axis as set by the annotation control routine ggSetAxesAnnotation() (ie, **xory** = GYAXIS):

ggSetAxesAnnotation(ndp,npower,axty,GYAXIS)

By default, values are output with up to two decimal places, but **ndp** may be set to adjust this if required. The use of ggSetAxesAnnotation() for Text Charts is to enable values output in the chart to match values displayed on the Y axis of a graph or chart. But it should be noted that **axty** has no effect on Text Charts and if **npower** is non-zero no scale factor is output with the Text Chart even though the values are multiplied by the appropriate power of 10. Again this can be used to match axes output if required.

All the values output by a call to ggDisplayValueColumn() and ggDisplayGeneratedColumn() may by augmented by a common prefix and/or suffix string of up to 30 characters as defined by the utility routine:

ggSetValueTags(prefix,suffix)

By default, both strings are null and if either string needs to be set to null, the string `*.' should be used as shown in the example below. Values output by ggDisplayPercentageColumn() are always followed by a `%' sign and cannot have a prefix or suffix string appended to them.

	Data	%
1	1.5mm	2.13%
2	3.0mm	4.26%
3	2.1mm	2.98%
4	4.4mm	6.24%
5	56.3mm	79.86%
6	3.2mm	4.54%

Multi-column value text chart

The following example shows the use of the three output routines and the two controlling routines in generating a multi-column Text Chart.

C Code

```
/*  MULTI-COLUMN VALUE TEXT CHART */
#include <gino-c.h>
#include <graf-c.h>

int main(void)
   int papty, nval;
   GDIM paper;
   float cw, ch, colwid, cellh, colhig, xp, yp;
   float values[6] = {1.5,3.0,2.1,4.4,56.3,3.2};

   gOpenGino();
   xxxxx();
   gEnqDrawingLimits(&paper,&papty);
   ggSetGraphCharMode(GGINOMODE);
```

```
/* SET UP TEXT CHART ATTRIBUTES */
   nval = 6;
   cw = 4.0;
   ch = 4.0;
   gSetCharSize(cw,ch);
   colwid = 7.0*cw;
   cellh = 2.0*ch;
   colhig = nval*cellh;

   ggSetTextChartAttribs(colwid,colhig,GCENTRE,GHEAD,1);
   xp =0.1*paper.xpap;
   yp =0.9*paper.ypap;
   ggDisplayGeneratedColumn(xp,yp,6,1.0,6.0," “);

/* CHANGE TEXT JUSTIFICATION FOR ACTUAL DATA */
   ggSetTextChartAttribs(colwid,colhig,GRIGHT,GHEAD,1);

/* ADD SUFFIX STRING */
   ggSetValueTags(”*.","mm");

/* SET NUMBER FORMAT TO FORCE ONE DECIMAL PLACE */
   ggSetAxesAnnotation(-1,0,GNOSCALE,GYAXIS);
   xp=xp+colwid;
   ggDisplayValueColumn(xp,yp,nval,values,"Data");
   xp=xp+colwid;
   ggDisplayPercentageColumn(xp,yp,nval,values,"%");

   gSuspendDevice();
   gCloseGino();
   return(0);
}
```

F90 Code

```
!  MULTI-COLUMN VALUE TEXT CHART
use gino_f90
use graf_f90

  integer papty, nval
  type (GDIM) paper
  real cw, ch, colwid, cellh, colhig, xp, yp
  real, dimension(6) :: values = (/1.5,3.0,2.1,4.4,56.3,3.2/)

  call gOpenGino
  call xxxxx
  call gEnqDrawingLimits(paper,papty)
  call ggSetGraphCharMode(GGINOMODE)

! SET UP TEXT CHART ATTRIBUTES
  nval = 6
  cw = 4.0
  ch = 4.0
  call gSetCharSize(cw,ch)
  colwid = 7.0*cw
  cellh = 2.0*ch
  colhig = nval*cellh

  call ggSetTextChartAttribs(colwid,colhig,GCENTRE,GHEAD,1)
  xp =0.1*xpap
  yp =0.9*ypap
  call ggDisplayGeneratedColumn(xp,yp,6,1.0,6.0,‘ ‘)
```

```
        ! CHANGE TEXT JUSTIFICATION FOR ACTUAL DATA
          call ggSetTextChartAttribs(colwid,colhig,GRIGHT,GHEAD,1)

        ! ADD SUFFIX STRING
          call ggSetValueTags('*.','mm')

        ! SET NUMBER FORMAT TO FORCE ONE DECIMAL PLACE
          call ggSetAxesAnnotation(-1,0,GNOSCALE,GYAXIS)
          xp=xp+colwid
          call ggDisplayValueColumn(xp,yp,nval,values,'data')
          xp=xp+colwid
          call ggDisplayPercentageColumn(xp,yp,nval,values,'%')

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Graphic Item Text Charts

Three routines are provided to link the graphical output on other graph forms with multi-column Text Charts:

ggDisplayMarkerColumn(x,y,nsym,sym,line,header)

ggDisplayLineColumn(x,y,nline,line,ang,header)

ggDisplayFillColumn(x,y,nfill,fill,line,header)

ggDisplayMarkerColumn() outputs a column of different symbols; ggDisplayLineColumn() outputs a column of lines drawn at various angles; and ggDisplayFillColumn() outputs a column of small rectangles filled with different fill styles. Each routine has an array of line styles (**line**) with which the item is drawn or filled. In each case the position **x**,**y** is the top left corner of the column and an optional header cell and the string **header** is added if headers are switched on with ggSetTextChartAttribs().

Where an application is presenting multiple data sets using graphs with different line styles or symbols, ggDisplayLineColumn() and ggDisplayMarkerColumn() can be simply used to annotate the different sets. Similarly, the fill style and line style arrays used with any of the chart or pie chart routines can be passed directly to ggDisplayFillColumn() again to annotate the data.

The fill styles and line styles used in these routines are set using the GINO line attribute routines.

The following example shows the various forms of output from these routines.

C Code

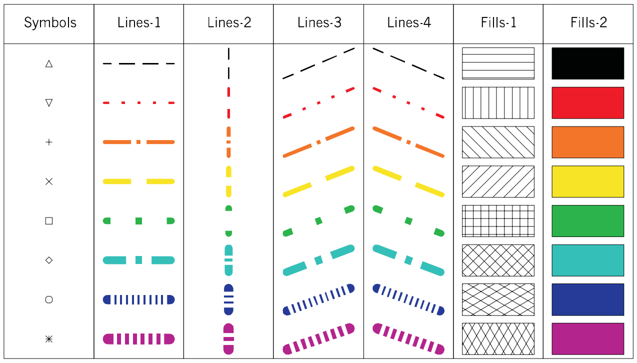
```
/*  GRAPHIC ITEM CHART */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
   int papty, nval, ncol, I;
   float colwid, colhig, cw, xp, yp;
   GDIM paper;
   GLIMIT lims;
   GLINSTY linrep;
   int line[8], line1[8], fill[8], sym[8];

   gOpenGino();

/* CALCULATE REQUIRED CHARACTER SIZE */
   cw = colwid/9.0;
   gSetCharSize(cw,cw);
   linrep.vis=GVISIBLE;
   xxxxx();
   gEnqDrawingLimits(&paper,&papty);
   gSetBrokenLineMode(GSOFT);
   lims.xmin=0.0;
   lims.xmax=paper.xpap;
   lims.ymin=0.0;
   lims.ymax=paper.ypap;
   gFillRect(GHOLLOW,GCURRENT,&paper);
   ggSetGraphCharMode(GGINOMODE);

/* SET UP TEXT CHART SIZE */
   ncol = 7;
   nval = 8;

/* CALCULATE COLUMN WIDTH AND HEIGHT */
   colwid = (0.8*paper.xpap)/(float)ncol;
   colhig = (0.6*paper.ypap);
   linrep.brk=GSOLID;
   linrep.col=GBLACK;
   linrep.width=0.0;
   linrep.type=GDEFAULT;
   linrep.end=GNONE;
   gDefineLineStyle(9,&linrep);
   ggSetTextChartAttribs(colwid,colhig,GCENTRE,GHEAD,9);

/* SET UP FILL,LINE AND SYMBOL NUMBERS */
   for (i=0; i<nval; i++) {
     line[i]=i;    line1[i]=1;
     fill[i]=i;
     sym[i]=i;
   }

/* DISPLAY CHARTS */
   xp =0.1*paper.xpap;
   yp =0.8*paper.ypap;
   ggDisplayMarkerColumn(xp,yp,nval,&sym,&line1,”Symbols”);
   xp = xp + colwid;
```

```
        /* SET UP LINE STYLES FOR LINE CHARTS - BROKEN, THICK AND ROUND ENDS
        */
           linrep.vis=GVISIBLE;
           linrep.type=GDEFAULT;
           linrep.end=GROUND;
           for (i=0; i<nval; i++) {
             linrep.brk=i;
             linrep.col=i;
             linrep.width=(float)i/3.0;
             gDefineLineStyle(i,&linrep);
           }
           ggDisplayLineColumn(xp,yp,nval,line,GHORIZONTAL,"lines-1");
           xp = xp + colwid;
           ggDisplayLineColumn(xp,yp,nval,line,GVERTICAL,"lines-2");
           xp = xp + colwid;
           ggDisplayLineColumn(xp,yp,nval,line,GRIGHTDIAGONAL,"lines-3");
           xp = xp + colwid;
           ggDisplayLineColumn(xp,yp,nval,line,GLEFTDIAGONAL,"lines-4");
           xp = xp + colwid;

        /* RESET LINE STYLES FOR FILL CHART */
           linrep.vis=GVISIBLE;
           linrep.type=GDEFAULT;
           linrep.end=GROUND;
           linrep.brk=GSOLID;
           linrep.width=0.0;
           for (i=0; i<nval; i++) {
             linrep.col=i;
             gDefineLineStyle(i,&linrep);
           }
           ggDisplayFillColumn(xp,yp,nval,&fill,&line1,”fills-1");
           xp = xp + colwid;

           for (i=0; i<nval; i++) {
             fill[i]=0;
           }
           ggDisplayFillColumn(xp,yp,nval,&fill,&line,“fills-2");

           gSuspendDevice();
           gCloseGino();
           return(0);
        }
```

F90 Code

```
        !  GRAPHIC ITEM CHART
        use gino_f90
        use graf_f90


          integer papty, nval, ncol, i
          real colwid, colhig, cw, xp, yp
          type (GDIM) paper
          type (GLIMIT) lims
          type (GLINSTY) linrep
          integer line(8), line1(8), fill(8), sym(8)
          call gOpenGino
          xxxxx
          call gEnqDrawingLimits(paper,papty)
          call gSetBrokenLineMode(GSOFT)
          lims%xmin=0.0
          lims%xmax=paper%xpap
          lims%ymin=0.0
          lims%ymax=paper%ypap
          call gFillRect(GHOLLOW,GCURRENT,paper)
          call ggSetGraphCharMode(GGINOMODE)

        ! SET UP TEXT CHART SIZE
          ncol = 7
          nval = 8

        ! CALCULATE COLUMN WIDTH AND HEIGHT
          colwid = (0.8*paper%xpap)/(float)ncol
          colhig = (0.6*paper%ypap)

        ! CALCULATE REQUIRED CHARACTER SIZE
          cw = colwid/9.0
          call gSetCharSize(cw,cw)
          linrep%vis=GVISIBLE
          linrep%brk=GSOLID
          linrep%col=GBLACK
          linrep%width=0.0
          linrep%type=GDEFAULT
          linrep%end=GNONE
          call gDefineLineStyle(9,linrep)
          call ggSetTextChartAttribs(colwid,colhig,GCENTRE, &
            GHEAD,9)

        ! SET UP FILL,LINE AND SYMBOL NUMBERS
          do i=1,nval
            line(i)=i
            line1(i)=1
            fill(i)=i
            sym(i)=i
          end do

        ! DISPLAY CHARTS
          xp =0.1*paper%xpap
          yp =0.8*paper%ypap
          call ggDisplayMarkerColumn(xp,yp,nval,sym,line1,'Symbols')
          xp = xp + colwid

        ! SET UP LINE STYLES FOR LINE CHARTS - BROKEN, THICK AND ROUND ENDS
          linrep%vis=GVISIBLE
          linrep%type=GDEFAULT
          linrep%end=GROUND

          do i=1,nval
            linrep%brk=i
            linrep%col=i
            linrep%width=(float)i/3.0
            call gDefineLineStyle(i,linrep)
          end do

          call ggDisplayLineColumn(xp,yp,nval,line,GHORIZONTAL,'lines-1')
          xp = xp + colwid
          call ggDisplayLineColumn(xp,yp,nval,line,GVERTICAL,'lines-2')
          xp = xp + colwid
          call ggDisplayLineColumn(xp,yp,nval,line,GRIGHTDIAGONAL,'lines-3')
```

```
      xp = xp + colwid
      call ggDisplayLineColumn(xp,yp,nval,line,GLEFTDIAGONAL,'lines-4')
      xp = xp + colwid

    ! RESET LINE STYLES FOR FILL CHART
      linrep%vis=GVISIBLE
      linrep%type=GDEFAULT
      linrep%end=GROUND
      linrep%brk=GSOLID
      linrep%width=0.0
      do i=1,nval
        linrep%col=i
        call gDefineLineStyle(i,linrep)
      end do
      call ggDisplayFillColumn(xp,yp,nval,fill,line1, &
        'fills-1')
      xp = xp + colwid
      do i=1,nval
        fill(i)=0
      end do
      call ggDisplayFillColumn(xp,yp,nval,fill,line, &
        'fills-2')

      call gSuspendDevice
      call gCloseGino
      stop
      end
```



Graphic Item Chart

Chapter 9

UTILITIES

Introduction to Utilities

There are several utility routines supplied by GINOGRAF to provide useful enhancements to the diagrams. They perform the following routines:

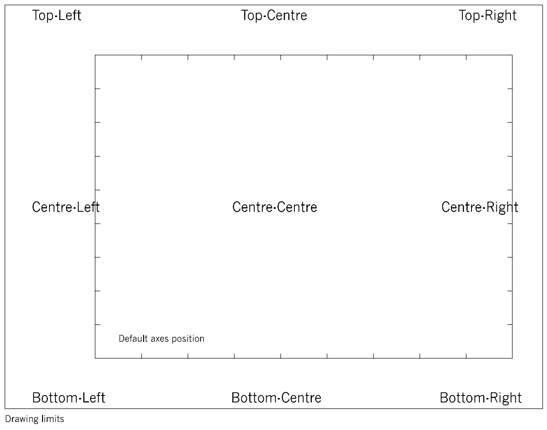
`ggDrawGraphTitle()`	Graph titling
`ggTransformGraphPoint()`	Graph to space coordinate conversion
`ggTransformSpacePoint()`	Space to graph coordinate conversion
`ggMoveToGraphPoint()`	Positioning
`ggAddGraphLine()`	Line drawing
`ggDrawArrow()`	Arrow drawing
`ggAddReferenceLine()`	Reference lines
`ggReturnLineCoeffs()`	Line Fitting

Graph Titling

The routine ggDrawGraphTitle() displays a title string at one of nine positions within the current graph drawing limits:

ggDrawGraphTitle(string,xpos,ypos)

This routine is provided to simplify the titling of graphs or charts where axes are positioned centrally within the drawing area. It does not check for any form of clashing with the graph or chart already drawn. **xpos** and **ypos** specify a justification in both the horizontal and vertical directions giving the following title positions with respect to the current GINOGRAF drawing limits.



Graph Title Positions

The title strings are output using the current GINO character and font settings.

Where ggDrawGraphTitle() does not provide enough positioning control it may be necessary to use the character and string routines within GINO to generate a suitable graph title. These are described in the Character Section of the GINO User Guide.

Coordinate Conversion

GINOGRAF provides two coordinate conversion routines. These convert from graphical axes coordinates to user space coordinates and vice versa.

ggTransformGraphPoint() converts a point from graphical axes coordinates (defined by ggSetAxesPos() and ggSetAxesScaling() or by one of the axis dependent high level routines) to user space coordinates as follows:

ggTransformGraphPoint(xgr,ygr,<u>point</u>)

xgr and **ygr** are the X and Y parts of the graphical axes coordinates to be converted. The user space coordinates are returned in **point.x** and **point.y**.

ggTransformSpacePoint() converts a point from user space coordinates to graphical axes coordinates as follows:

ggTransformSpacePoint(xsp,ysp,<u>point</u>)

xsp and **ysp** are the X and Y parts of the user space coordinates to be converted. The graphical axes coordinates are returned in **point.x** and **point.y**.

Line Drawing

There are two aspects to drawing a line:

- Positioning the pen
- Drawing the line

ggMoveToGraphPoint() positions the pen with respect to the graphical axes coordinates as follows:

ggMoveToGraphPoint(x,y)

x and **y** are the coordinates of a point in the coordinate system defined by ggSetAxesPos() and ggSetAxesScaling() or by one of the axis dependent high level routines. The point need not be within the area defined by the axes. The pen is moved to the position (**x**, **y**) without drawing a line.

ggAddGraphLine() draws a line with respect to the graphical axes coordinates as follows:

ggAddGraphLine(x,y)

ggAddGraphLine() draws a line from the current pen position set up by ggMoveToGraphPoint() to the position (**x**,**y**).

Arrow Drawing

There are two aspects to drawing an arrow:

- Positioning the pen
- Drawing the arrow

If graphical axes are being used, the GINOGRAF routine ggMoveToGraphPoint() can be used to position the pen with respect to the graphical axes coordinates as follows:

```
ggMoveToGraphPoint(x,y)
```

x and **y** are coordinates of a point in the coordinate system defined by ggSetAxesPos() and ggSetAxesScaling() or one of the axis-dependent high level routines. The point need not be within the area defined by the axes.

The pen is moved to the position (**x**, **y**) without drawing a line. If pie chart or text chart routines are being used, the GINO routine gMoveTo2D() can be used to position the pen in user space coordinates as follows:

```
gMoveTo2D(xsp,ysp)
```

xsp and **ysp** are the user space coordinates of the point. The pen is moved to the position (**xsp**, **ysp**) without drawing a line.

ggDrawArrow() draws the line and the arrowhead as follows:

```
ggDrawArrow(xhead, yhead, head, mode)
```

ggDrawArrow() draws a line from the current pen position (set up by ggMoveToGraphPoint() or gMoveTo2D()) to the position (**xhead**, **yhead**) and draws an arrowhead there. The position can be specified either in graphical coordinates (with **mode** = GGRAPH) or in user space coordinates (with **mode** = GSPACE).

head specifies the type of arrowhead. If **head** = GOPEN, the arrowhead is open and if **head** = GCLOSED, it is closed. If **head** = GSOLID, the arrowhead is filled using solid colour in the current GINO colour.

Reference Lines

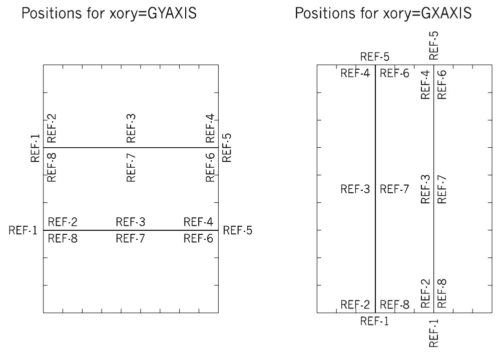
A reference or target value may be indicated on a graph using ggAddReferenceLine() which draws a labelled line horizontally or vertically across a set of axes.

```
ggAddReferenceLine(string,xyval,labjus,labclock,hv,xory)
```

The routine draws a line at a value indicated by **xyval**, on either the X or Y axis across the full length of the opposite axis. **xyval** is measured in graphical axes coordinates, according to the current axes position and scaling set by ggSetAxesPos() and ggSetAxesScaling(). If the value selected lies outside of the axes limits then the reference line is not drawn.

A text string may be output in one of eight positions according to the value of **labjus** and **labclock** around the reference line as shown below:

	labjus	labclock
REF-1	GFARLEFT	NA
REF-2	GLEFT	GANTICLOCKWISE
REF-3	GCENTRE	GANTICLOCKWISE
REF-4	GRIGHT	GANTICLOCKWISE
REF-5	GFARRIGHT	NA
REF-6	GRIGHT	GCLOCKWISE
REF-7	GCENTRE	GCLOCKWISE
REF-8	GLEFT	GCLOCKWISE



Reference line annotation positions

The orientation of the text string can be parallel with the X or Y axes using **hv**, where **hv**=GXAXIS displays the text parallel with the X axes, and **hv**=GYAXIS displays the text parallel with the Y axes as shown above.

Data Fitting

The routine ggReturnLineCoeffs() provides some basic data fitting facilities:

ggReturnLineCoeffs(type,npts,points,ncoef,coeffs,nmax,<u>err</u>)

where **npts** data points are provided in **points.x** and **points.y** and the routine returns the corresponding coefficients of the fitted line or curve in **coeffs**. No graphics output is generated by this routine.

The required number of coefficients is set in **ncoef** which should also be the size of the array **coeffs** in which they are placed. **nmax** returns the actual number of coefficients calculated which may be more or less than **ncoef** depending on the algorithm. **err** is set to GSUCCESS if a successful fit has been made. If a fit is not possible **err** is set to GFAIL and no coefficients are returned.

At present only least squares straight line fitting is provided (**type**=GLEASTSQUARE) and so only two coefficients are calculated; `a’ and `b’ where the line y=ax + b best fits the supplied data.

Example Program

The following example program shows usage of most of the above utility routines together with its output:

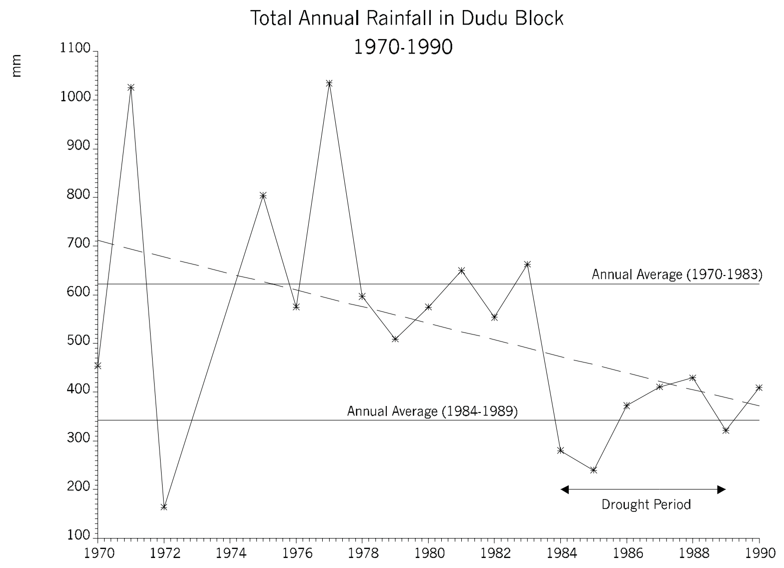
C Code

```
/*  EXAMPLE PROGRAM USING UTILITY ROUTINES */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
#define NPT 19
   int i,papty,nc,er;
   float ch, co[2], y1, y2;
   GLIMIT lims; GDIM paper;
   GPOINT spcpnt;
   GPOINT pnts[NPT] = {70.0,454.3,71.0,1025.9,
     72.0,163.8,75.0,804.2,76.0,575.0,77.0,1035.0,
     78.0,597.0,79.0,509.0,80.0,575.0,81.0,650.0,
     82.0,554.0,83.0,662.8,84.0,280.0,85.0,239.6,
     86.0,372.3,87.0,410.8,88.0,430.0,89.0,321.2,
     90.0,409.8};

   for (i=0; i<NPT; i++) {
     pnts[i].x=pnts[i].x+1900;
   }

   gOpenGino();
   xxxxx();
   gEnqDrawingLimits(&paper,&papty);
   ch=2.0;
   gSetCharSize(ch,ch);
   ggSetGraphCharMode(GGINOMODE);

/* SET NEW DRAWING LIMITS */
   lims.xmin=0.1*paper.xpap;
   lims.xmax=0.9*paper.xpap;
   lims.ymin=0.1*paper.ypap;
   lims.ymax=0.9*paper.ypap;
   ggSetPlotFrame(&lims);

/* DRAW GRAPH */
   ggPlotGraph(NPT,pnts,GLINEAR,GLINEAR,-GSTRAIGHT,GAXES);
   ggDrawAxesTitle(“mm”,0.1*paper.ypap,GYAXIS,GTOP,GRIGHT);
/* FIT DATA AND DRAW TREND LINE */
   ggReturnLineCoeffs(GLEASTSQUARE,NPT,pnts,2,co,&nc,&er);
   y1=co[1]*1970.0+co[0];
   y2=co[1]*1990.0+co[0];
   gSetBrokenLine(GSHORTDASHED);
   ggMoveToGraphPoint(1970.0,y1);
   ggAddGraphLine(1990.0,y2);
   gSetBrokenLine(GSOLID);

/* ADD REFERENCE LINE */
   ggAddReferenceLine("Annual Average (1970-1984)",623.0,
       GRIGHT,GANTICLOCKWISE,GXAXIS,GYAXIS);
   ggAddReferenceLine("Annual Average (1984-1989)",342.0,
       GCENTRE,GANTICLOCKWISE,GXAXIS,GYAXIS);

/* ADD ARROW AND TEXT */
   ggMoveToGraphPoint(1987.0,200.0);
   ggDrawArrow(1984.0,200.0,GSOLID,GGRAPH);
   ggMoveToGraphPoint(1987.0,200.0);
   ggDrawArrow(1989.0,200.0,GSOLID,GGRAPH);

/* CONVERT GRAPHICAL COORDINATES INTO SPACE COORDINATES */
   ggTransformGraphPoint(1986.5,200.0,&spcpnt);
   gMoveTo2D(spcpnt.x,spcpnt.y);
   gSetStrJustify(GCENTRE);
   gDisplayStr("Drought Period");

/* ADD GRAPH TITLE */
   ch=3.0;
   gSetCharSize(ch,ch);
   ggDrawGraphTitle
       ("Total Annual Rainfall in Dudu Block*N1970-1990",
        GCENTRE,GTOP);
   gSuspendDevice();
}
```

F90 Code

```
! EXAMPLE PROGRAM USING UTILITY ROUTINES
use gino_f90
use graf_f90

  integer, parameter :: NPT = 19
  integer papty, nc, er
  real ch, co(2), y1, y2
  type (GLIMIT) lims
  type (GDIM) paper
  type (GPOINT) spcpnt
  type (GPOINT), dimension(NPT) :: pnts = &
    (/GPOINT(70.0,454.3),GPOINT(71.0,1025.9), &
      GPOINT(72.0,163.8),GPOINT(75.0,804.2), &
      GPOINT(76.0,575.0),GPOINT(77.0,1035.0), &
      GPOINT(78.0,597.0),GPOINT(79.0,509.0), &
      GPOINT(80.0,575.0),GPOINT(81.0,650.0), &
      GPOINT(82.0,554.0),GPOINT(83.0,662.8), &
      GPOINT(84.0,280.0),GPOINT(85.0,239.6), &
      GPOINT(86.0,372.3),GPOINT(87.0,410.8), &
      GPOINT(88.0,430.0),GPOINT(89.0,321.2), &
      GPOINT(90.0,409.8)/)
```

```
          do i=1,NPT
            pnts(i)%x=pnts(i)%x+1900
          end do

          call gOpenGino
          call xxxxx
          call gEnqDrawingLimits(paper,papty)
          ch=2.0
          call gSetCharSize(ch,ch)
          call ggSetGraphCharMode(GGINOMODE)

        ! SET NEW DRAWING LIMITS
          lims%xmin=0.1*paper%xpap
          lims%xmax=0.9*paper%xpap
          lims%ymin=0.1*paper%ypap
          lims%ymax=0.9*paper%ypap
          call ggSetPlotFrame(lims)

        ! DRAW GRAPH
          call ggPlotGraph(NPT,pnts,GLINEAR,GLINEAR,-GSTRAIGHT, &
            GAXES)
          call ggDrawAxesTitle('mm',0.1*paper%ypap,GYAXIS,GTOP, &
            GRIGHT)
        ! FIT DATA AND DRAW TREND LINE
          call ggReturnLineCoeffs(GLEASTSQUARE,NPT,pnts, &
            2,co,nc,er)
          y1=co(2)*1970.0+co(1)
          y2=co(2)*1990.0+co(1)
          call gSetBrokenLine(GSHORTDASHED)
          call ggMoveToGraphPoint(1970.0,y1)
          call ggAddGraphLine(1990.0,y2)
          call gSetBrokenLine(GSOLID)
        ! ADD REFERENCE LINE
          call ggAddReferenceLine('Annual Average (1970-1984)', &
                 623.0,GRIGHT,GANTICLOCKWISE,GXAXIS,GYAXIS)
          call ggAddReferenceLine('Annual Average (1984-1989)',  &
                 342.0,GCENTRE,GANTICLOCKWISE,GXAXIS,GYAXIS)

        ! ADD ARROW AND TEXT
          call ggMoveToGraphPoint(1987.0,200.0)
          call ggDrawArrow(1984.0,200.0,GSOLID,GGRAPH)
          call ggMoveToGraphPoint(1987.0,200.0)
          call ggDrawArrow(1989.0,200.0,GSOLID,GGRAPH)

        ! CONVERT GRAPHICAL COORDINATES INTO SPACE COORDINATES
          call ggTransformGraphPoint(1986.5,200.0,spcpnt)
          call gMoveTo2D(spcpnt%x,spcpnt%y)
          call gSetStrJustify(GCENTRE)
          call gDisplayStr('Drought Period')

        ! ADD GRAPH TITLE
          ch=3.0
          call gSetCharSize(ch,ch)
          call ggDrawGraphTitle &
               ('Total Annual Rainfall in Dudu Block*N1970-1990', &
                GCENTRE,GTOP)
          call gSuspendDevice
          stop
        end
```



Example output using utility routines

Chapter 10

GRAPH LAYOUT

Introduction to Graph Layout

Most of the examples in the previous chapters of this manual show a single data representation on a single set of axes. It is of course possible with GINOGRAF to place multiple graphs or charts on one page, place multiple data sets on a single set of axes and multiple data sets on multiple sets of axes. This chapter does not introduce any new routines but describes how to obtain complex graphical output by means of sample coding.

Multiple Graph Layout

There are two ways in which to layout more than one graph onto a single sheet of paper or screen:

Using Complete Graph or Chart Routines

If the Complete Graph or Chart drawing routines are being used, different graph drawing areas can be defined using the routine ggSetPlotFrame(). These can be as proportions of paper/screen dimensions as shown below or using physical dimensions as appropriate. The Complete Drawing routine automatically calculates the size and position of the graph or chart to fit within the new area.

C Code

```
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GLIMIT lims;
  GDIM paper;
  int papty;
```

```
          gOpenGino();
          xxxxx();
          gEnqDrawingLimits(&paper,&papty)

          lims.xmin=0.0;
          lims.xmax=0.4*paper.xpap;
          lims.ymin=0.0;
          lims.ymax=paper.ypap;
          ggSetPlotFrame(&lims);
          :
          ...Use of Complete Graph or Chart Routine...
          :
          lims.xmin=0.5*paper.xpap;
          lims.xmax=0.9*paper.xpap;
          ggSetPlotFrame(&lims);
          :
          ...Use of Complete Graph or Chart Routine...
          :
          gSuspendDevice();
          gCloseGino();
          return 0;
        }
```

F90 Code

```
        use gino_f90
        use graf_f90


          type (GLIMIT) lims
          type (GDIM) paper
          integer papty

          call gOpenGino
          call xxxxx
          call gEnqDrawingLimits(paper,papty)

          lims%xmin=0.0
          lims%xmax=0.4*paper%xpap
          lims%ymin=0.0
          lims%ymax=paper%ypap
          call ggSetPlotFrame(lims)
          :
          ...Use of Complete Graph or Chart Routine...
          :
          lims%xmin=0.5*paper%xpap
          lims%xmax=0.9*paper%xpap
          call ggSetPlotFrame(lims)
          :
          ...Use of Complete Graph or Chart Routine...
          :
          call gSuspendDevice
          call gCloseGino
          stop
          end
```

A complete example using ggSetPlotFrame() can be seen in the code that accompanies the chapter entitled Pie Chart Annotation control . Remember, that it may be necessary to reduce the default GINO character size when laying out multiple graphs as the current dimensions are used in the layout calculations, if not for purely aesthetic reasons.

Using Component Routines

If the low-level component routines are being used, the relevant axes or chart position routines are used with the appropriate positions, lengths and/or radii. For example, ggSetAxesPos() would be used for normal graphs and charts, ggSetPolarChartAttribs() for polar charts and ggSetPieChartFrame() for pie charts. Again proportions of paper/screen dimensions or physical dimensions can be used as appropriate.

C Code

```
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GLIMIT lims;
  GDIM paper;
  int papty;

  gOpenGino();
  xxxxx();
  gEnqDrawingLimits(&paper,&papty);

/* POSITION AXES IN RIGHT HALF OF PAGE */
  ggSetAxesPos(GAXISSTART,0.1*paper.xpap,
    0.1*paper.ypap,0.4*paper.xpap,GXAXIS);
  ggSetAxesPos(GAXISSTART,0.1*paper.xpap,
    0.1*paper.ypap,0.8*paper.ypap,GYAXIS);
    :
/* POSITION AXES IN LEFT HALF OF PAGE */
  ggSetAxesPos(GAXISSTART,0.6*paper.xpap,
    0.1*paper.ypap,0.4*paper.xpap,GXAXIS);
  ggSetAxesPos(GAXISSTART,0.6*paper.xpap,
    0.1*paper.ypap,0.8*paper.ypap,GYAXIS);
      :
  gSuspendDevice();
  gCloseGino();
  return 0;
}
```

F90 Code

```
        use gino_f90
        use graf_f90

          type (GLIMIT) lims
          type (GDIM) paper
          integer papty

          call gOpenGino
          call xxxxx
          call gEnqDrawingLimits(paper,papty)

        ! POSITION AXES IN RIGHT HALF OF PAGE
          call ggSetAxesPos(GAXISSTART,0.1*paper.xpap, &
            0.1*paper.ypap,0.4*paper.xpap,GXAXIS)
          call ggSetAxesPos(GAXISSTART,0.1*paper.xpap, &
            0.1*paper.ypap,0.8*paper.ypap,GYAXIS)
            :
        ! POSITION AXES IN LEFT HALF OF PAGE
          call ggSetAxesPos(GAXISSTART,0.6*paper.xpap, &
            0.1*paper.ypap,0.4*paper.xpap,GXAXIS)
          call ggSetAxesPos(GAXISSTART,0.6*paper.xpap, &
            0.1*paper.ypap,0.8*paper.ypap,GYAXIS)
              :
          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Graphs with Multiple Data Sets

Once both axes have been defined and the graphical axes coordinate system has been set up, there is no limit on the number of data sets that may be drawn in that system.

Mixing Graph Levels

The availability of the Complete Graph or Chart routines provide a quick and easy route to obtain a graph or chart with the minimum of effort. As explained in the relevant chapters, the Complete Drawing routines use the default axes positions and calculate appropriate axes ranges to include all the supplied data. Once a Complete Drawing routine has been used a graphical coordinate system has been set up and any of the component drawing routines may be used on the same set of axes.

For example, after a graph has been drawn using ggPlotGraph(), error bars may be added with ggAddErrorBars() and data values annotated with ggAddGraphValues(), or after a chart outline has been drawn with ggPlotBarChart(), the same bars may be filled with ggFillBarChart().

However, when mixing a Complete Drawing routine with Component routines, the following points should be noted:

- If additional data sets are to be added after using a Complete Drawing routine, ensure that the data set with the largest data range was supplied to that routine. This is because the Complete Drawing routine calculates the data limits of the data set it was supplied with and sets up the axes ranges to match. Therefore any data set added to these axes should lie within these limits.
- If filling is done after a Complete Drawing routine, Graph and Chart outlines will be overdrawn by the filling. If the filling extends to the axis, the line of the axis may also be overdrawn. In such circumstances, the Graph or Chart boundary or the relevant axis may be redrawn if required.

Mixing Graph Types

If both X and Y axes are defined as having continuous scaling types (linear or logarithmic), there is no restriction on using any of the graph forms, Step or Area Chart forms on those axes.

If one of the axes has been defined with discrete scaling as required for Histograms and Bar Charts, there is also no restriction on adding any other Graph or Chart form except in its interpretation.

If both axes are defined as having continuous scaling and a Histogram or Bar Chart is added, an error message is output. The chart is still output, however, with the major tick marks of the X axis assumed to be the discrete intervals.

If Polar Charts are to be used in conjunction with normal graphs, it should be realised that the Polar Chart drawing routines will redefine the position and scaling of graphical axes as set up with the routines ggSetAxesPos() and ggSetAxesScaling(). If particular settings need to be saved and restored, the axis enquiry routines ggEnqAxesPos() and ggEnqAxesScaling() can be used.

Graphs with Multiple Axes

As the GINOGRAF user has complete control over the position and scaling of graphical axes it is possible to layout a graph with multiple axes in many different forms. These include:

- Multiple axes on graph frame
- Axes with multiple scales
- Conversion scales

Examples of each of the above are given in the following code examples:

Multiple Axes on Graph Frame

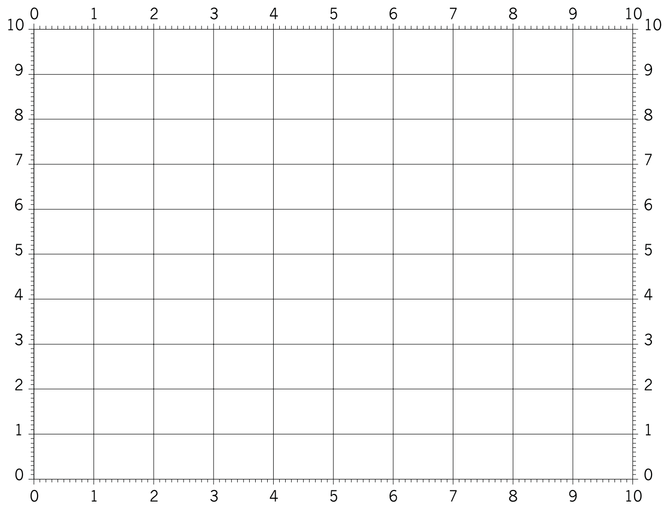
The default axis frame format, using the routine ggAddGrid(), only annotates the bottom and left axes. If all four sides of the frame require annotation it is necessary to use the individual axis positioning, scaling and drawing routines (see page 19). The example below positions the new frame centrally within the paper/screen area whereas the routine ggAddGrid() uses the default axis positions.

C Code

```
/*  GRAPH FRAME WITH MULTIPLE AXES */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GDIM paper;
  int papty;

  gOpenGino();
  xxxxx();
  gEnqDrawingLimits(&paper,&papty);
  ggSetGraphCharMode(GGINOMODE);

/* POSITION AND SCALE LEFT AXIS */
  ggSetAxesPos(GAXISSTART,0.1*paper.xpap,
    0.1*paper.ypap,0.8*paper.ypap,GYAXIS);
  ggSetAxesScaling(GLINEARTYPE3,10,0.0,10.0,GYAXIS);
  ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE,
    GANTICLOCKWISE,GYAXIS);
/* POSITION AND SCALE LOWER AXIS */
  ggSetAxesPos(GAXISSTART,0.1*paper.xpap,
    0.1*paper.ypap,0.8*paper.xpap,GXAXIS);
  ggSetAxesScaling(GLINEARTYPE3,10,0.0,10.0,GXAXIS);
  ggDrawAxes(GINTERMEDIATE,GCLOCKWISE,GCLOCKWISE,
    GXAXIS);
/* POSITION AND SCALE RIGHT AXIS */
  ggSetAxesPos(GAXISSTART,0.9*paper.xpap,
    0.1*paper.ypap,0.8*paper.ypap,GYAXIS);
  ggSetAxesScaling(GLINEARTYPE3,10,0.0,10.0,GYAXIS);
  ggDrawAxes(GINTERMEDIATE,GCLOCKWISE,GCLOCKWISE,
    GYAXIS);
/* POSITION AND SCALE UPPER AXIS */
  ggSetAxesPos(GAXISSTART,0.1*paper.xpap,
    0.9*paper.ypap,0.8*paper.xpap,GXAXIS);
  ggSetAxesScaling(GLINEARTYPE3,10,0.0,10.0,GXAXIS);
  ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE,
    GANTICLOCKWISE,GXAXIS);

/* DRAW INTERNAL GRID WITH NO ANNOTATION */
  ggAddGrid(GCARDINAL,GGRIDLINES,GNOANNOTATION,
    GNOANNOTATION);

  gSuspendDevice();
  gCloseGino();
  return 0;
}
```

F90 Code

```
!  GRAPH FRAME WITH MULTIPLE AXES
use gino_f90
use graf_f90


  type (GDIM) paper
  integer papty

  call gOpenGino
  call xxxxx
  call gEnqDrawingLimits(paper,papty)
  call ggSetGraphCharMode(GGINOMODE)

! POSITION AND SCALE LEFT AXIS
  call ggSetAxesPos(GAXISSTART,0.1*paper%xpap, &
    0.1*paper%ypap,0.8*paper%ypap,GYAXIS)
  call ggSetAxesScaling(GLINEARTYPE3,10,0.0,10.0,GYAXIS)
  call ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE, &
    GANTICLOCKWISE,GYAXIS)
! POSITION AND SCALE LOWER AXIS
  call ggSetAxesPos(GAXISSTART,0.1*paper%xpap, &
    0.1*paper%ypap,0.8*paper%xpap,GXAXIS)
  call ggSetAxesScaling(GLINEARTYPE3,10,0.0,10.0,GXAXIS)
  call ggDrawAxes(GINTERMEDIATE,GCLOCKWISE,GCLOCKWISE, &
    GXAXIS)
! POSITION AND SCALE RIGHT AXIS
  call ggSetAxesPos(GAXISSTART,0.9*paper%xpap, &
    0.1*paper%ypap,0.8*paper%ypap,GYAXIS)
  call ggSetAxesScaling(GLINEARTYPE3,10,0.0,10.0,GYAXIS)
  call ggDrawAxes(GINTERMEDIATE,GCLOCKWISE,GCLOCKWISE, &
    GYAXIS)
! POSITION AND SCALE UPPER AXIS
  call ggSetAxesPos(GAXISSTART,0.1*paper%xpap, &
    0.9*paper%ypap,0.8*paper%xpap,GXAXIS)
  call ggSetAxesScaling(GLINEARTYPE3,10,0.0,10.0,GXAXIS)
  call ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE, &
    GANTICLOCKWISE,GXAXIS)

! DRAW INTERNAL GRID WITH NO ANNOTATION
  call ggAddGrid(GCARDINAL,GGRIDLINES,GNOANNOTATION, &
    GNOANNOTATION)

  call gSuspendDevice
  call gCloseGino
  stop
  end
```



Graph Frame with Multiple Axes

Axes with Multiple Scales

On many occasions it is necessary to display multiple data sets which have one common axis but require different ranges or scales on the other axis. As it is possible in GINOGRAF to define axes scales separately it is possible to define the X axis once and subsequently define the Y axis several times for the different data sets. The point to note is that any data set will be displayed with reference to the last call to ggSetAxesPos()/ggSetAxesScaling(), therefore a program which requires multiple scales must set the appropriate axis scaling before displaying the relevant data set.

The different scales for the Y axis may be displayed in one of three ways:

- At different heights
- Adjacent axes
- On the same physical axis

In the first and second cases, both the position (ggSetAxesPos()) and scale (ggSetAxesScaling()) is defined for each data set. In the third case two different scales can be displayed on different sides of the same axis (using ggDrawAxes()), therefore the position is defined once but the scale is defined for each data set. Obviously, where the different data sets occupy the same drawing area, as with the second and third cases, it is more important to label the data sets appropriately.

Note that an axis may be positioned anywhere on the drawing area as long as it is in line with the other axis without affecting the graphical axes coordinate system it defines. In other words, a Y axis may be placed at any X coordinate on the screen or paper, as long as the base is at the correct Y position in relation to the X axis position.

The example below shows the three cases of multiple axes definitions.

C Code

```
/*  VARIOUS LAYOUTS OF GRAPHS WITH MULTIPLE
   SCALES */
#include <gino-c.h>
#include <graf-c.h>

#define NPTS 10
void timeax(GDIM paper,float txp,float yp,int n);
void dtemp(GDIM paper,float xp,float yp,float *time,
  float *temp,int n,int tick,int tickside);
void dpres(GDIM paper,float xp,float yp,float *time,
  float *pres,int n,int tick,int tickside);

int main(void) {
  float time[NPTS],
    temp[NPTS]={0.0,5.0,11.0,15.0,25.0,28.0,
      31.0,32.0,33.0,33.0},
    pres[NPTS]={82.0,80.0,78.0,75.0,71.0,66.0,
      60.0,55.0,54.0,51.0};
  GDIM paper;
  int i,papty;

  for (i=0; i<NPTS; i++) {
    time[i]=(float)(i+1);
  }

void timeax(GDIM paper,float xp,float yp,int n) {
/* POSITION, SCALE AND DRAW TIME AXIS */

  ggSetAxesPos(GAXISSTART,xp,yp,0.4*paper.xpap,
    GXAXIS);
  ggSetAxesScaling(GLINEARTYPE3,n-1,1.0,(float)n,
    GXAXIS);
  ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS);
  ggDrawAxesTitle("Time",yp-5.0,GXAXIS,GTOP,GRIGHT);
}
```

```
    gOpenGino();
    xxxxx();
    gEnqDrawingLimits(&paper,&papty);
    gSetCharSize(2.0,2.0);
    ggRestoreAxesSettings();
    ggSetGraphCharMode(GGINOMODE);

  /* DIVIDE AREA INTO THREE */
    gMoveTo2D(0.5*paper.xpap,0.0);
    gDrawLineBy2D(0.0,paper.ypap);
    gMoveTo2D(0.5*paper.xpap,0.5*paper.ypap);
    gDrawLineBy2D(0.5*paper.xpap,0.0);

   /* 1) DATA SETS PLACED AT DIFFERENT HEIGHTS */
    timeax(paper,0.05*paper.xpap,0.1*paper.ypap,NPTS);
    dtemp(paper,0.05*paper.xpap,0.2*paper.ypap,time,temp,
      NPTS,GCARDINAL,GANTICLOCKWISE);
    dpres(paper,0.05*paper.xpap,0.6*paper.ypap,time,pres,
      NPTS,GCARDINAL,GANTICLOCKWISE);
    ggDrawAxesTitle("separate axes",0.95*paper.ypap,
      GXAXIS,GTOP,GCENTRE);

  /* 2) SINGLE AXIS WITH MULTIPLE SCALES */
    timeax(paper,0.55*paper.xpap,0.6*paper.ypap,NPTS);
    dtemp(paper,0.55*paper.xpap,0.6*paper.ypap,time,temp,
      NPTS,GCARDINAL,GANTICLOCKWISE);
    dpres(paper,0.55*paper.xpap,0.6*paper.ypap,time,pres,
      NPTS,GCARDINAL,GCLOCKWISE);
    ggDrawAxesTitle("composite axis",0.95*paper.ypap,
      GXAXIS,GTOP,GCENTRE);

  /* 3) ADJACENT AXES */
    timeax(paper,0.55*paper.xpap,0.1*paper.ypap,NPTS);
    dtemp(paper,0.95*paper.xpap,0.1*paper.ypap,time,temp,
      NPTS,GCARDINAL,GCLOCKWISE);
    dpres(paper,0.55*paper.xpap,0.1*paper.ypap,time,pres,
      NPTS,GCARDINAL,GANTICLOCKWISE);
    ggDrawAxesTitle("adjacent axes",0.45*paper.ypap,
      GXAXIS,GTOP,GCENTRE);

    gSuspendDevice();
    gCloseGino();
    return 0;
  }
```

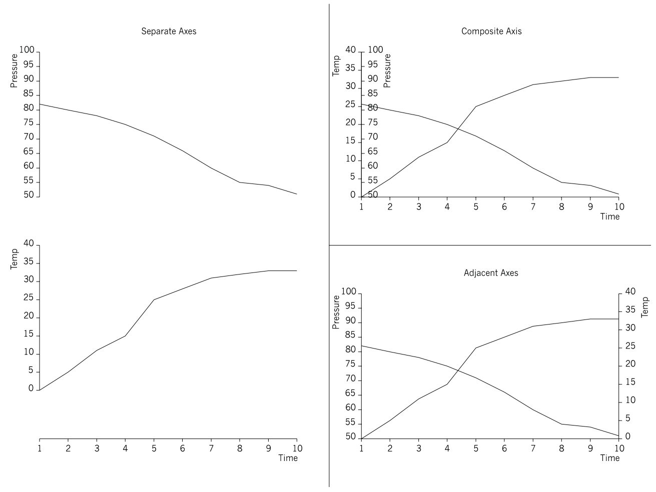
```
void dtemp(GDIM paper,float xp,float yp,float *time,
  float *temp,int n,int tick,int tickside) {
/* POSITION, SCALE AND DRAW TEMPERATURE AXIS */
/* DRAW AND LABEL DATA SET */

  int i, side;
  float xpa;
  GPOINT pline[NPTS];

  for (i=0; i<NPTS; i++) {
    pline[i].x=time[i];
    pline[i].y=temp[i];
  }
  ggSetAxesPos(GAXISSTART,xp,yp,
    0.3*paper.ypap,GYAXIS);
  ggSetAxesScaling(GLINEARTYPE3,8,0.0,40.0,GYAXIS);
  ggDrawAxes(tick,tickside,tickside,GYAXIS);
  if (tickside==GANTICLOCKWISE) {
    xpa=xp-7.0;
    side=GBOTTOM;
  } else {
    xpa=xp+7.0;
    side=GTOP;
  }
  ggDrawAxesTitle("Temp",xpa,GYAXIS,side,GTOP);
  ggAddGraphPolyline(n,pline);
}
void dpres(GDIM paper,float xp,float yp,float *time,
  float *pres,int n,int tick,int tickside) {
/* POSITION, SCALE AND DRAW PRESSURE AXIS */
/* DRAW AND LABEL DATA SET */
  int i, side;
  float xpa;
  GPOINT pline[NPTS];

  for (i=0; i<NPTS; i++) {
    pline[i].x=time[i];
    pline[i].y=pres[i];
  }
  ggSetAxesPos(GAXISSTART,xp,yp,
    0.3*paper.ypap,GYAXIS);
  ggSetAxesScaling(GLINEARTYPE3,10,50.0,100.0,GYAXIS);
  ggDrawAxes(tick,tickside,tickside,GYAXIS);
  if (tickside==GANTICLOCKWISE) {
    xpa=xp-7.0;
    side=GBOTTOM;
  } else {
    xpa=xp+7.0;
    side=GTOP;
  }
  ggDrawAxesTitle("Pressure",xpa,GYAXIS,side,GTOP);
  ggAddGraphPolyline(n,pline);
}
```

F90 Code

```
!  VARIOUS LAYOUTS OF GRAPHS WITH MULTIPLE
! SCALES
use gino_f90
use graf_f90

  parameter (NPTS=10)
  real time(NPTS)
  real :: temp(NPTS)=(/0.0,5.0,11.0,15.0,25.0,28.0, &
    31.0,32.0,33.0,33.0/)
  real :: pres(NPTS)=(/82.0,80.0,78.0,75.0,71.0,66.0, &
    60.0,55.0,54.0,51.0/)
  type (GDIM) paper
  integer i,papty

  do i=1,NPTS
    time(i)=real(i)
  end do

  call gOpenGino
  xxxxx
  call gEnqDrawingLimits(paper,papty)
  call gSetCharSize(2.0,2.0)
  call ggRestoreAxesSettings
  call ggSetGraphCharMode(GGINOMODE)

! DIVIDE AREA INTO THREE
  call gMoveTo2D(0.5*paper%xpap,0.0)
  call gDrawLineBy2D(0.0,paper%ypap)
  call gMoveTo2D(0.5*paper%xpap,0.5*paper%ypap)
  call gDrawLineBy2D(0.5*paper%xpap,0.0)

! 1) DATA SETS PLACED AT DIFFERENT HEIGHTS
  call timeax(paper,0.05*paper%xpap,0.1*paper%ypap,NPTS)
  call dtemp(paper,0.05*paper%xpap,0.2*paper%ypap,time,temp, &
    NPTS,GCARDINAL,GANTICLOCKWISE)
  call dpres(paper,0.05*paper%xpap,0.6*paper%ypap,time,pres, &
    NPTS,GCARDINAL,GANTICLOCKWISE)
  call ggDrawAxesTitle(‘separate axes’,0.95*paper%ypap, &
    GXAXIS,GTOP,GCENTRE)

! 2) SINGLE AXIS WITH MULTIPLE SCALES
  call timeax(paper,0.55*paper%xpap,0.6*paper%ypap,NPTS)
  call dtemp(paper,0.55*paper%xpap,0.6*paper%ypap,time,temp, &
    NPTS,GCARDINAL,GANTICLOCKWISE)
  call dpres(paper,0.55*paper%xpap,0.6*paper%ypap,time,pres, &
    NPTS,GCARDINAL,GCLOCKWISE)
  call ggDrawAxesTitle(‘composite axis’,0.95*paper%ypap, &
    GXAXIS,GTOP,GCENTRE)

! 3) ADJACENT AXES
  call timeax(paper,0.55*paper%xpap,0.1*paper%ypap,NPTS)
  call dtemp(paper,0.95*paper%xpap,0.1*paper%ypap,time,temp, &
    NPTS,GCARDINAL,GCLOCKWISE)
  call dpres(paper,0.55*paper%xpap,0.1*paper%ypap,time,pres, &
    NPTS,GCARDINAL,GANTICLOCKWISE)
  call ggDrawAxesTitle(‘adjacent axes’,0.45*paper%ypap, &
    GXAXIS,GTOP,GCENTRE)
```

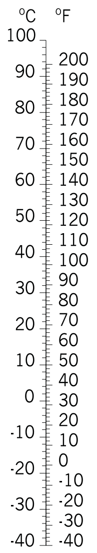
```
          call gSuspendDevice
          call gCloseGino
          stop
          end

        subroutine timeax(paper,xp,yp,n)
        ! POSITION, SCALE AND DRAW TIME AXIS
        use gino_f90
        use graf_f90
          parameter (NPTS=10)
          type (GDIM) paper
          real xp,yp
          integer n

          call ggSetAxesPos(GAXISSTART,xp,yp,0.4*paper%xpap, &
            GXAXIS)
          call ggSetAxesScaling(GLINEARTYPE3,n-1,1.0,real(n), &
            GXAXIS)
          call ggDrawAxes(GCARDINAL,GCLOCKWISE,GCLOCKWISE,GXAXIS)
          call ggDrawAxesTitle('Time',yp-5.0,GXAXIS,GTOP,GRIGHT)
          return
          end

        subroutine dtemp(paper,xp,yp,time,temp,n,tick,tickside)
        ! POSITION, SCALE AND DRAW TEMPERATURE AXIS
        ! DRAW AND LABEL DATA SET
        use gino_f90
        use graf_f90
          parameter (NPTS=10)
          type (GDIM) paper
          real xp,yp
          real, dimension(NPTS) :: time, temp
          integer i,side,n,tick,tickside
          real xpa
          type (GPOINT), dimension(NPTS) :: pline

          do i=1,NPTS
            pline(i)%x=time(i)
            pline(i)%y=temp(i)
          end do
          call ggSetAxesPos(GAXISSTART,xp,yp, &
            0.3*paper%ypap,GYAXIS)
          call ggSetAxesScaling(GLINEARTYPE3,8,0.0,40.0,GYAXIS)
          call ggDrawAxes(tick,tickside,tickside,GYAXIS)
          if(tickside==GANTICLOCKWISE) then
            xpa=xp-7.0
            side=GBOTTOM
          else
            xpa=xp+7.0
            side=GTOP
          end if
          call ggDrawAxesTitle('Temp',xpa,GYAXIS,side,GTOP)
          call ggAddGraphPolyline(n,pline)
          return
          end
```

```
subroutine dpres(paper,xp,yp,time,pres,n,tick,tickside)
! POSITION, SCALE AND DRAW PRESSURE AXIS
! DRAW AND LABEL DATA SET
use gino_f90
use graf_f90
  parameter (NPTS=10)
  type (GDIM) paper
  real xp,yp
  real, dimension(NPTS) :: time
  real, dimension(NPTS) :: pres
  integer i,side,n,tick,tickside
  real xpa
  type (GPOINT), dimension(NPTS) :: pline

  do i=1,NPTS
    pline(i)%x=time(i)
    pline(i)%y=pres(i)
  end do
  call ggSetAxesPos(GAXISSTART,xp,yp, &
    0.3*paper%ypap,GYAXIS)
  call ggSetAxesScaling(GLINEARTYPE3,10,50.0,100.0,GYAXIS)
  call ggDrawAxes(tick,tickside,tickside,GYAXIS)
  if(tickside==GANTICLOCKWISE) then
    xpa=xp-7.0
    side=GBOTTOM
  else
    xpa=xp+7.0
    side=GTOP
  endif
  call ggDrawAxesTitle('Pressure',xpa,GYAXIS,side,GTOP)
  call ggAddGraphPolyline(n,pline)
  return
  end
```



Various layouts of multiple scale axis

Conversion Scales



Celsius/Fahrenheit Conversion Scale

A special case of different scales on the same physical axis is where precise and accurate association of the ranges and appropriate annotation is required as in the case of conversion scales. Special consideration is required in this situation because ggSetAxesScaling() does not provide a suitable scaling type which will 'nicely' annotate an axis given a precise range. The example below shows the steps required to display a suitably annotated temperature conversion scale on the same axis. In this example it is necessary to define a different length for the second axis in order to obtain the required annotation.

C Code

```
/*  FAHRENHEIT/CENTIGRADE CONVERSION SCALE */
#include <gino-c.h>
#include <graf-c.h>

int main(void) {
  GDIM paper;
  int papty;
  float axlen,zerof,topf;
  GPOINT pnt;

  gOpenGino();
  xxxxx();
  gEnqDrawingLimits(&paper,&papty);
  ggSetGraphCharMode(GGINOMODE);

/* POSITION CENTIGRADE AXES */
  ggSetAxesPos(GDATAORIGIN,0.5*paper.xpap,
    0.3*paper.ypap,0.8*paper.ypap,GYAXIS);

/* SET UP CENTIGRADE SCALE FOR Y AXIS AND ANNOTATE ON
   ANTI-CLOCKWISE SIDE */
  ggSetAxesScaling(GLINEARTYPE3,14,-40.0,100.0,
    GYAXIS);
  ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE,
    GANTICLOCKWISE,GYAXIS);
  gSetStrJustify(GRIGHT);
  gMoveTo2D(0.5*paper.xpap,0.9*paper.ypap);
  gDisplayStr("*Eo*AC *.");

/* CALCULATE ORIGIN OF FAHRENHEIT AXIS */
  zerof=-32.0*5.0/9.0;
  ggTransformGraphPoint(0.0,zerof,&pnt);
/* CALCULATE LENGTH OF FAHRENHEIT AXES
   (FROM -40 TO 200) */
  topf=(200.0-32.0)*5.0/9.0;
  axlen=0.8*paper.ypap*(topf+40.0)/140.0;

/* POSITION FAHRENHEIT AXES WITH NEW CALCULATED LENGTH */
  ggSetAxesPos(GDATAORIGIN,0.5*paper.xpap,
    pnt.y,axlen,GYAXIS);

/* POSITION FAHRENHEIT AXES WITH NEW CALCULATED LENGTH */
  ggSetAxesPos(GDATAORIGIN,0.5*paper.xpap,
    pnt.y,axlen,GYAXIS);

/* SET UP SECOND SCALE AND ANNOTATE ON CLOCKWISE SIDE */
  ggSetAxesScaling(GLINEARTYPE3,24,-40.0,200.0,
    GYAXIS);
  ggDrawAxes(GINTERMEDIATE,GCLOCKWISE,GCLOCKWISE,
    GYAXIS);
  gSetStrJustify(GLEFT);
  gMoveTo2D(0.5*paper.xpap,0.9*paper.ypap);
  gDisplayStr(" *Eo*AF*.");

  gSuspendDevice();
  gCloseGino();
  return 0;
}
```

F90 Code

```
        !  FAHRENHEIT/CENTIGRADE CONVERSION SCALE
        use gino_f90
        use graf_f90

          type (GDIM) paper
          integer papty
          real axlen,zerof,topf
          type (GPOINT) pnt

          call gOpenGino
          call xxxxx
          call gEnqDrawingLimits(paper,papty)
          call ggSetGraphCharMode(GGINOMODE)

        ! POSITION CENTIGRADE AXES
          call ggSetAxesPos(GDATAORIGIN,0.5*paper%xpap, &
          0.3*paper%ypap,0.8*paper%ypap,GYAXIS)

        ! SET UP CENTIGRADE SCALE FOR Y AXIS AND ANNOTATE ON
        ! ANTI-CLOCKWISE SIDE
          call ggSetAxesScaling(GLINEARTYPE3,14,-40.0,100.0, &
            GYAXIS)
          call ggDrawAxes(GINTERMEDIATE,GANTICLOCKWISE, &
            GANTICLOCKWISE,GYAXIS)
          call gSetStrJustify(GRIGHT)
          call gMoveTo2D(0.5*paper%xpap,0.9*paper%ypap)
          call gDisplayStr('*Eo*AC *.')

        ! CALCULATE ORIGIN OF FAHRENHEIT AXIS
          zerof=-32.0*5.0/9.0
          call ggTransformGraphPoint(0.0,zerof,pnt)
        ! CALCULATE LENGTH OF FAHRENHEIT AXES
        ! (FROM -40 TO 200)
          topf=(200.0-32.0)*5.0/9.0
          axlen=0.8*paper%ypap*(topf+40.0)/140.0

        ! POSITION FAHRENHEIT AXES WITH NEW CALCULATED LENGTH
          call ggSetAxesPos(GDATAORIGIN,0.5*paper%xpap, &
            pnt%y,axlen,GYAXIS)

        ! POSITION FAHRENHEIT AXES WITH NEW CALCULATED LENGTH
          call ggSetAxesPos(GDATAORIGIN,0.5*paper%xpap, &
            pnt%y,axlen,GYAXIS)

        ! SET UP SECOND SCALE AND ANNOTATE ON CLOCKWISE SIDE
          call ggSetAxesScaling(GLINEARTYPE3,24,-40.0,200.0, &
            GYAXIS)
          call ggDrawAxes(GINTERMEDIATE,GCLOCKWISE,GCLOCKWISE, &
            GYAXIS)
          call gSetStrJustify(GLEFT)
          call gMoveTo2D(0.5*paper%xpap,0.9*paper%ypap)
          call gDisplayStr(' *Eo*AF*.')

          call gSuspendDevice
          call gCloseGino
          stop
          end
```

Chapter 11

ROUTINE SPECIFICATIONS

An Introduction to Routine Specifications

This chapter of the manual includes a detailed description of all the available routines in GINOGRAF listed alphabetically.

Where return variables can be used for arguments that are changeable within a routine, these are indicated by being underlined thus.

Each routine specification provides the following information:

- An example call showing the specification of the routine
- A definition of each of the arguments
- A description of the task the routine performs
- A reference to the relevant pages of this manual (and to any other appropriate documentation) which discuss the use of the routine

ggAddAkimaCurve

Syntax

C/C++:	**void ggAddAkimaCurve**(int npts, GPOINT *points);

F90:	**subroutine ggAddAkimaCurve**(npts, points) integer, intent(in) :: npts type(GPOINT), intent(in) :: points(*)

Arguments

npts
The number of points to be plotted

points
Array of points through which a curve is to be drawn on the graph

Description

The routine ggAddAkimaCurve() draws a smooth Akima curve through the points defined in array **points**, with respect to the current axes as set up by the last axis definition calls. The points are not marked on the graph. If axes have not been defined, the points will be read and plotted according to the default values for axis positioning and scaling. This could result in points being outside the available drawing area or current window.

The end conditions of the curve may be set using the routines ggSetCurveStartConds() and/or ggSetCurveEndConds().

See Also

Page 54
ggSetCurveStartConds
ggSetCurveEndConds

ggAddAreaChartOutline

Syntax

C/C++:	**void ggAddAreaChartOutline**(int nareas,GAREACHART *areas,int xory);

F90:	**subroutine ggAddAreaChartOutline**(nareas, areas, xory) integer, intent(in) :: nareas, xory type(GAREACHART), intent(in) :: areas(*)

Arguments

nareas
The number of areas to be plotted

areas
Array of structures giving data for the Area Chart

xory
Specified axis on which the data widths are represented

= GXAXIS	Data on X axis, heights on Y axis
= GYAXIS	Data on Y axis, heights on X axis

Description

The routine ggAddAreaChartOutline() draws rectangles defined by the data held in the array of structures **areas**. The four components represent the start and finish widths and the lower and upper heights with respect to either the current axes as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default axes used by one of the high level routines. The widths of the areas, held in **areas.s** and **areas.f**, are shown on the X-axis if **xory**=GXAXIS, and on the Y-axis if **xory**=GYAXIS.

See Also

Page 108
ggPlotAreaChart
ggFillAreaChart
ggAddAreaChartValues

ggAddAreaChartValues

Syntax

```
C/C++:    void ggAddAreaChartValues(int nareas, GAREACHART *areas, int sfl, int xory);
```

```
F90:      subroutine ggAddAreaChartValues(nareas, areas, sfl, xory)

          integer, intent(in) :: nareas, sfl, xory
          type(GAREACHART), intent(in) :: areas(*)
```

Arguments

nareas
The number of areas to be annotated

areas
Array of structures giving data for the Area Chart

sfl
Flag determining which aspect of each area is output

= GSTART	Value of **areas.s** is output
= GFINISH	Value of **areas.f** is output
= GLOWER	Value of **areas.h1** is output
= GUPPER	Value of **areas.h2** is output
= GWIDTH	Area width is output (ie, **areas.f** - **areas.s**)
= GHEIGHT	Area height is output (ie, **areas.h2** - **areas.h1**)
= GAREA	Area of rectangle is output (ie, (**areas.f-areas.s**)*(**areas.h2-areas.h1**))

xory
Specified axis on which the data widths are represented

= GXAXIS	X axis
= GYAXIS	Y axis

Description

The routine ggAddAreaChartValues() displays the area coordinates, widths, heights, or areas, with respect to the current axes as set up by the last axis definition calls or through a previous call to ggPlotAreaChart(). **xory** determines which direction the areas are displayed, thus if **xory**=GXAXIS the **areas.s** and **areas.f** values are displayed on the X axis and **areas.h1** and **areas.h2** values are displayed on the Y axis. If ggAddAreaChartValues() follows a call to ggPlotAreaChart(), **xory** should be set to GXAXIS.

The most recent call to ggSetValueAttribs() determines the position of the annotation about each area as well as the angle, justification and offset of the value string. The default is to position the required value at the centre of each area. Start, finish, width and area values are output according to the format of the **xory** axis, whereas lower, upper and area height values are output according to the format of the other axis. The format is set by the most recent call to ggSetAxesAnnotation(). Prefix and suffix strings defined by the most recent call to ggSetValueTags() are appended to the numerical output.

If **sfl** is out or range, a warning message is output and the area of the rectangles are displayed.

See Also

Page 109
ggSetAxesAnnotation
ggPlotAreaChart
ggFillAreaChart
ggAddAreaChartOutline
ggSetValueAttribs
ggSetValueTags

ggAddBarChartOutline

Syntax

```
C/C++:    void ggAddBarChartOutline(int nbars,GBARCHART *bars, float frac);
```

```
F90:      subroutine ggAddBarChartOutline(nbars, bars, frac)

          type(GBARCHART), intent(in) :: bars(*)
          integer, intent(in) :: nbars
          real, intent(in) :: frac
```

Arguments

nbars
The number of bars to be plotted

bars
Array of structures giving data for the Bar Chart outline

frac
Value between 0.0 and 1.0 inclusive which specifies the fraction of an interval to be occupied by each bar. If **frac** = 1.0, only the necessary lines are drawn, ie, lines common to two bars are omitted. If **frac** = 0.0, two coincident lines are drawn centred on the tick mark

Description

The routine ggAddBarChartOutline() draws a Bar Chart of **nbars** bars defined in array of structures **bars**, where **bars.s** and **bars.f** represent the start and finish values with respect to the current axes as set up by the last axis definition calls. The bars on the Bar Chart have the width ((length of discrete axis)/**nbars**) * **frac**.

The bars are centred on the tick marks on the discrete axis. If a discrete axis has not been defined using ggSetAxesScaling() (ie, **scale** = GDISCRETE), or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

See Also

Page 96
ggSetAxesScaling
ggPlotBarChart
ggFillBarChart
ggAddBarChartValues

ggAddBarChartValues

Syntax

```
C/C++:    void ggAddBarChartValues(int nbars, GBARCHART *bars, float frac, int sfl);
```

```
F90:      subroutine ggAddBarChartValues(nbars, bars, frac, sfl)

          type(GBARCHART), intent(in) :: bars(*)
          integer, intent(in) :: nbars, sfl
          real, intent(in) :: frac
```

Arguments

nbars

The number of bars

bars

Array of structures giving data for the Bar Chart values

frac

Value between 0.0 and 1.0 inclusive which specifies the fraction of the interval occupied by each column for compatibility with Bar Chart outline

sfl

Flag determining which value is to be output

= GSTART	Start value
= GFINISH	Finish value
= GLENGTH	Length of bar

Description

The routine ggAddBarChartValues() annotates a Bar Chart data set with start, finish or length values as defined in array of structures **bars**, with respect to the current axes. The routine requires one of the axes to be defined as a discrete axis. This can be done by using ggSetAxesScaling() (with **scale**=GDISCRETE) or through a previous call to ggPlotBarChart(). If no discrete axis has been defined or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

The most recent call to ggSetValueAttribs() determines the position of the annotation about each bar as well as the angle, justification and offset of the value string. The default is to position the required value at the centre of each bar. The format of the numerical output is determined by the settings for the current non-discrete axis annotation set by the most recent call to ggSetAxesAnnotation(). Prefix and suffix strings defined by the most recent call to ggSetValueTags() are appended to the numerical output.

If **sfl** is out of range, a warning message is output and the length of the bar is displayed.

See Also

Page 97
ggSetAxesAnnotation
ggSetAxesScaling
ggPlotBarChart
ggSetValueAttribs
ggSetValueTags

ggAddErrorBars

Syntax

```
C/C++:    void ggAddErrorBars(int npts, GPOINT *points, GERROR *errors, int type, int line,
          int xory);
```

```
F90:      subroutine ggAddErrorBars(npts, points, errors, type, line, xory)

          integer, intent(in) :: npts, type, line, xory
          type(GPOINT), intent(in) :: points(*)
          type(GERROR), intent(in) :: errors(*)
```

Arguments

npts
The number of error bars to be plotted

points
Array of data points to be defined on the graph

errors
Array of relative error values below and above each corresponding value on the graph

type
Error bar symbol end type

= GNONE	No end
= GBAR	Horizontal bar
= GARROWSIN	Arrows pointing inwards
= GTRIANGLESIN	Triangles pointing inwards
= GSOLIDTRIANGLESIN	Filled triangles pointing inwards
= GARROWSOUT	Arrows pointing outwards
= GTRIANGLESOUT	Triangles pointing outwards
= GSOLIDTRIANGLESOUT	Filled triangles pointing outwards

line
Flag determining whether or not a line is drawn between error limits

= GNONE	No line
= GDRAWLINE	Line is drawn

xory
Specified axis

= GXAXIS	Error bars shown perpendicular to X axis
= GYAXIS	Error bars shown perpendicular to Y axis

Description

The routine ggAddErrorBars() plots an error bar at each of the points on a graph within the defined axes. The bars are displayed as absolute distances above and below the values stored in the array **points**. The relative distances above the values are stored in **error.upper**, the relative distances below are stored in **error.lower**. The ends of the error bars may be displayed in one of eight different forms depending on the value of **type**. The symbol size being subject to the current GINO character width.

The bar ends may be joined by a line depending on the value of **line**.

If axes have not been defined, the points will be read and plotted according to the default values for axis positioning and scaling.

See Also

Page 57

ggAddGraphCurve

Syntax

```
C/C++:        void ggAddGraphCurve(int npts, GPOINT *points);
```

```
F90:          subroutine ggAddGraphCurve(npts, points)

              integer, intent(in) :: npts
              type(GPOINT), intent(in) :: points(*)
```

Arguments

npts
The number of points to be plotted

points
Array of points through which a curve is to be drawn on the graph

Description

The routine ggAddGraphCurve() draws a smooth curve through the points defined in array **points**, with respect to the current axes as set up by the last axis definition calls. The points are not marked on the graph. If axes have not been defined, the points will be read and plotted according to the default values for axis positioning and scaling. This could result in points being outside the available drawing area or current window.

The end conditions of the curve may be set using the routines ggSetCurveStartConds() and/or ggSetCurveEndConds().

See Also

Page 54
ggSetCurveStartConds
ggSetCurveEndConds

ggAddGraphLine

Syntax

```
C/C++:        void ggAddGraphLine(float x, float y);
```

```
F90:          subroutines ggAddGraphLine(x, y)
              real, intent(in) :: x, y
```

Arguments

x
Value giving the X part of the graphical axes coordinate to which the line will be drawn

y
Value giving the Y part of the graphical axes coordinate to which the line will be drawn

Description

The routine ggAddGraphLine() draws a line from the current pen position to the point (**x**, **y**), either inside or outside the graph limits, in the graphical axes system set up by the last axis definition calls or by one of the axis control routines.

See Also

Page 167
ggMoveToGraphPoint

ggAddGraphMarkers

Syntax

```
C/C++:   void ggAddGraphMarkers(int npts, GPOINT *points, int sym, int nspace);
```

```
F90:     subroutine ggAddGraphMarkers(npts, points, sym, nspace)

         integer, intent(in) :: npts, sym, nspace
         type(GPOINT), intent(in) :: points(*)
```

Arguments

npts
The number of points to be plotted

points
Array of dimension **npts**, giving the X and Y axis values of the points to be defined on the graph

sym
Symbol number to be drawn

= GSPOT	}
= GUP	}
= GDOWN	}
= GPLUS	}
= GCROSS	} Standard GINO symbols
= GBOX	}
= GDIAMOND	}
= GCIRCLE	}
= GSTAR	}
= 9 to 23	Optional hardware symbol
> 23	Character from GINO font table

nspace
Integer, positive or zero, giving the number of points to be left unmarked between each point at which a symbol is drawn

Description

The routine ggAddGraphMarkers() plots a symbol at some or all of the points on a graph within the defined axes. ggAddGraphMarkers() can be used to superimpose symbols on straight line or smooth curve graphs.

The **npts** points defined in array **points** are considered, and the symbol **sym** is drawn at the first point. Each successive symbol is then drawn after an interval of **nspace** points, eg: if **nspace** = 3, symbols are drawn at points 1,5,9,13 etc. If **nspace** is negative, a default of 0 is assumed.

sym is the symbol number that will be drawn at each requested point. The standard nine symbols (0-8) that are available are shown in Appendix A. However, other symbols are available through access to hardware symbols or the GINO font file. Details of this is found in the GINO Manual, Appendix C and the routine gDrawMarker() in the Subroutine Specification chapter. The symbol size is subject to the current GINO character size. If **sym** is negative a default of GUP is assumed.

Graphs drawn by the routine ggAddGraphMarkers() can be affected by the current missing value mode as set by the routine ggDefineMissingValues().

See Also

Page 56
ggDefineMissingValues

ggAddGraphPolyline

Syntax

```
C/C++:      void ggAddGraphPolyline(int npts, GPOINT *points);
```

```
F90:        subroutine ggAddGraphPolyline(npts, points)
            integer, intent(in) :: npts
            type(GPOINT), intent(in) :: points
```

Arguments

npts

The number of points to be plotted

points

Array of dimension **npts**, giving the X and Y axis values of the points to be defined on the graph

Description

The routine ggAddGraphPolyline() draws straight line segments between the points defined in array **points**, with respect to the current axes as set up by the last axis definition calls. The points are not marked on the graph. If axes have not been defined, the points will be read and plotted according to the default values for axis positioning and scaling. This could result in points being outside the available drawing area or current window.

Graphs drawn by the routine ggAddGraphPolyline() can be affected by the current missing value mode as set by the routine ggDefineMissingValues().

See Also

Page 54
ggDefineMissingValues

ggAddGraphSpline

Syntax

```
C/C++:     void ggAddGraphSpline(int npts, GPOINT *points);
```

```
F90:       subroutine ggAddGraphSpline(npts, points)

           integer, intent(in) :: npts
           type(GPOINT), intent(in) :: points(*)
```

Arguments

npts
The number of points to be plotted

points
Array of dimension **npts**, giving the X and Y axis values of the points to be defined on the graph

Description

The routine ggAddGraphSpline() fits a cubic spline through the points defined in array **points**, with respect to the current axes as set up by the last axis definition calls. The points are not marked on the graph. If axes have not been defined, the points will be read and plotted according to the default values for axis positioning and scaling. This could result in points being outside the available drawing area or current window.

The end conditions may be set by using the routines ggSetCurveStartConds() and/or ggSetCurveEndConds().

See Also

Page 54
ggSetCurveStartConds
ggSetCurveEndConds

ggAddGraphValues

Syntax

```
C/C++:     void ggAddGraphValues(int npts, GPOINT *points, int xory);
```

```
F90:       subroutine ggAddGraphValues(npts, points, xory)

           integer, intent(in) :: npts, xory
           type(GPOINT), intent(in) :: points(*)
```

Arguments

npts
The number of points to be annotated

points
Array of dimension **npts**, giving the X and Y axis values of the points defined on the graph

xory
Flag determining whether X or Y values are output

= GXAXIS	X values are output
= GYAXIS	Y values are output

Description

The routine ggAddGraphValues() displays the values held in either the **points.x** or **points.y** components, depending on the value of **xory**.

The most recent call to ggSetValueAttribs() determines the angle, justification, offset, and the position of the annotation about the data points, the default being centrally over the point being annotated. ggSetValueAttribs() offers 16 different positions about each data point for the output.

The format of the numerical output is determined by the settings for the appropriate axis annotation set by the most recent call to ggSetAxesAnnotation(). Prefix and suffix strings defined by the most recent call to ggSetValueTags() are output either side of the numerical output.

Graphs drawn by the routine ggAddGraphValues() can be affected by the current missing value mode as set by the routine ggDefineMissingValues().

See Also

Page 64
ggDefineMissingValues
ggSetAxesAnnotation
ggSetValueAttribs
ggSetValueTags

ggAddGrid

Syntax

```
C/C++:        void ggAddGrid(int style1, int style2, int anx, int any);
```

```
F90:          subroutine ggAddGrid(style1, style2, anx, any)
              integer, intent(in) :: style1, style2, anx, any
```

Arguments

style1
The style of frame to be drawn

= GNONE	Draws a plain frame
= GCARDINAL	Draws a frame with markings at only the cardinal points
= GINTERMEDIATE	Draws a frame with markings at the cardinal and intermediate points

style2
The style of grid (and frame) to be drawn

= GNONE	Draws a plain frame
= GTICKS	Draws a frame with only tick marks
= GTICKSANDCROSSES	Draws a frame with tick marks and grid markers
= GGRIDLINES	Draws a grid made up of lines
= GGRIDLINES2	Draws a grid made up of lines but forces any intermediate lines to be drawn using the default line thickness

anx
Flag determining whether annotation and/or grid lines are drawn with the X axis

= GNOGRID	Suppresses X axis and grid lines perpendicular to the X axis
= GNOANNOTATION	Suppresses annotation on the X axis
= GANNOTATION	Annotation values written outside the frame on the X axis

any
Flag determining whether annotation and/or grid lines are drawn with the Y axis

= GNOGRID	Suppresses Y axis and grid lines perpendicular to the Y axis
= GNOANNOTATION	Suppresses annotation on the Y axis
= GANNOTATION	Annotation values written outside the frame on the Y axis

Description

The routine ggAddGrid() draws a frame and/or a grid for a Graph, Histogram or Bar Chart. Grid positioning and scaling are controlled by ggSetAxesPos() and ggSetAxesScaling(). No intermediate tick marks are drawn on a discrete axis, irrespective of the value of **style1** & **style2**. If either axis is discrete, tick marks on both axes are drawn outside the frame. X and/or Y zero coordinate lines are drawn if they occur within the data range and linear scaling is being used.

If either of **anx** or **any** is set to GNOANNOTATION, the annotation of the respective axis is suppressed. If either **anx** or **any** is set to GNOGRID, the annotation, major (and minor) tick marks are suppressed as well as the components of the grid lines perpendicular to that axis. The grid intersection symbols are unaffected.

The annotation is displayed with the attributes set with the most recent call to ggSetAxesAnnotation() and ggSetAxesAttribs(). The grid intersection marker for **style2** set to GTICKSANDCROSSES is set by the routine ggSetGridMarker(), the default being a small cross.

The grid is drawn using the current GINO line thickness and style except when **style1**=GINTERMEDIATE and **style2**=GGRIDLINES2 where the intermediate lines are drawn using the default line thickness.

See Also

Page 25
ggSetAxesPos
ggSetAxesScaling
ggSetAxesAnnotation
ggSetAxesAttribs
ggSetGridMarker

ggAddHistogramOutline

Syntax

```
C/C++:   void ggAddHistogramOutline(int ncols, float *yarray, float frac);
```

```
F90:     subroutine ggAddHistogramOutline(ncols, yarray, frac)

         integer, intent(in) :: ncols
         real, intent(in) :: yarray(*), frac
```

Arguments

ncols
The number of columns to be plotted

yarray
Array, of dimension **ncols**, giving the heights of the columns to be plotted on the Histogram

frac
Fraction of an interval to be occupied by each column

Description

The routine ggAddHistogramOutline() draws a Histogram of **ncols** columns, with heights defined in array **yarray**, with respect to the current axes, as set up by the last axis definition calls. The columns in the Histogram have the width:

((length of X axis)/**ncols**) * **frac**

If **frac** = 1.0, only the necessary verticals are drawn, ie, lines common to two columns are omitted. If **frac** = 0.0, two coincident vertical lines are drawn centred on the tick mark. If **frac** < 0.0, the default value is 0.0 and if **frac** > 1.0, the default value is 1.0.

The columns are centred on the tick marks on the discrete axis. If a discrete axis has not been defined using ggSetAxesScaling() (ie, **scale** = GDISCRETE), or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

If a height defined in array **yarray** is negative, the column is drawn on the negative side of the discrete axis.

See Also

Page 89
ggPlotHistogram
ggFillHistogram
ggAddHistogramValues

ggAddHistogramValues

Syntax

C/C++:	**void ggAddHistogramValues**(int ncols, float *yarray, float frac);

```
F90:        subroutine ggAddHistogramValues(ncols, yarray, frac)

            integer, intent(in) :: ncols
            real, intent(in) :: yarray(*), frac
```

Arguments

ncols
The number of columns in the Histogram

yarray
Array, of dimension **ncols**, giving the heights of all the columns in the Histogram

frac
Fraction of an interval occupied by each column

Description

The routine ggAddHistogramValues() annotates a Histogram with height values defined in array **yarray**, with respect to the current axes as set up by the last axis definition calls. ggAddHistogramValues() requires one of the axes to be defined as a discrete axis. This can be done by using ggSetAxesScaling() (with **scale**=GDISCRETE) or through a previous call to ggPlotHistogram(). If no discrete axis has been defined or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

The most recent call to ggSetValueAttribs() determines the position of the annotation about each column as well as the angle, justification and offset of the value string. The default is to position the required value at the centre of each column. Where Histogram data values are negative, the position of the control point on the non-discrete axis is placed in the matching position corresponding to the column height but below the zero axis.

The format of the numerical output is determined by the settings for the current non-discrete axis annotation set by the most recent call to ggSetAxesAnnotation(). Prefix and suffix strings defined by the most recent call to ggSetValueTags() are appended to the numerical output.

See Also

Page 91
ggSetAxesAnnotation
ggSetAxesScaling
ggPlotHistogram
ggSetValueAttribs
ggSetValueTags

ggAddPieChartSegment

Syntax

C/C++:	**void ggAddPieChartSegment**(float angfro, float angto, float value,char string[ ], int fill, int line);

F90:	**subroutine ggAddPieChartSegment**(angfro, angto, value, string, fill, line) real, intent(in) :: angfro, angto, value character*(*), intent(in) :: string integer, intent(in) :: fill, line

Arguments

angfro

Number giving the angle (in degrees), measured anticlockwise from the three o'clock position, which defines the start of the segment

angto

Number giving the angle (in degrees), measured anticlockwise from the three o'clock position, which defines the end of the segment

value

Data value associated with segment. The value need not be a percentage as ggAddPieChartSegment() automatically calculates the percentage of the whole pie chart that the segment occupies

string

Text string containing segment label

fill

Fill style to be used to fill the segment

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}

> 256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line

Integer determining the line style to be used to fill the segment. The value of **line** is irrelevant where **fill** has a value less than -1

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

Description

The routine ggAddPieChartSegment() draws a single annotated, filled Pie Chart segment anticlockwise between the angles **angfro** and **angto**, and with a radius and centre to fit the available drawing window or with respect to the Pie Chart frame defined by the most recent call to ggSetPieChartFrame().

The segment consists of the following elements; the background filling, the annotation and associated box and the segment boundary.

The segment is filled in the style determined by the combination of the fill and line style indices **fill** and **line**. Where **fill** is equal to GHOLLOW, only the boundary of the segment is drawn. The segment may be left unfilled by setting **fill** to -2 or less. Negative values of **line** cause a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

The default annotation for the Pie Chart is to print the segment label horizontally in a masked box within the segment boundary. Other forms of annotation are available including radial and external, each of which may include a combination of the segment label, the percentage value of the segment and the data value itself. All these options are set with the Pie Chart annotation routine ggSetPieChartAnnotation(). The routine ggSetPieChartBoxType() controls the filling/masking and drawing of the annotation box for internal segment annotation.

By default the segment boundary is drawn in the current pen colour, this can be switched off using ggSetPieChartBoundSwitch().

See Also

Page 142
ggSetPieChartStartAngle
ggSetPieChartAnnotation
ggSetPieChartBoxType
ggSetPieChartBoundSwitch
ggSetPieChartExplosion
ggSetPieChartFrame

ggAddPopulationGraph

Syntax

C/C++:

```
void ggAddPopulationGraph(float dx, float y, int npts, GPOINT *points, float popmax);
```

F90:

```
subroutine ggAddPopulationGraph(dx, y, npts, points, popmax)

real,              intent(in) :: dx, y, popmax
integer,           intent(in) :: npts
type(GPOINT), intent(in) :: points
```

Arguments

dx
Population sample interval

y
Population

npts
The number of points to be plotted

points
Array of dimension **npts**, giving the sample point and population counts to be defined on the graph

popmax
Maximum population

Description

The routine ggAddPopulationGraph() draws a double line population graph on the currently specified set of axes, one of which should be of a discrete scaling type using ggSetAxesScaling(). The graph represents a single set of population sample data (**y**) on the discrete axis which has a regular recording interval **dx** along the non-discrete axis.

The data suppled in the array **points** consist of a set of population recordings **points.y** at the specified sample point **points.x**. Missing recordings in the **points** array are represented by gaps in the double line graph.

The value **popmax** is supplied to control the scaling of the graph such that the range 0.0 - **popmax** lies between two points of the discrete axis (**y**).

See Also

Page 59
ggSetAxesScaling

ggAddReferenceLine

Syntax

C/C++:	**void ggAddReferenceLine**(char string[ ], float xyval, int labjus, int labclock, int hv, int xory);

F90:	**subroutine ggAddReferenceLine**(string, xyval, labjus, labclock, hv, xory) character*(*), intent(in) :: string real, intent(in) :: xyval integer, intent(in) :: labjus, labclock, hv, xory

Arguments

string
Variable or constant holding the title

xyval
X or Y axis intercept value in graphical axes coordinates

labjus
Position of text string around reference line

= GFARLEFT	Beyond lower limit, **labclock** is ignored
= GLEFT	Left justified at lower limit
= GCENTRE	Central
= GRIGHT	Right justified at upper limit
= GFARRIGHT	Beyond upper limit, **labclock** is ignored

labclock
Side of reference line to position text string

= GCLOCKWISE	Position the text string on the clockwise side of the reference line
= GANTICLOCKWISE	Position the text string on the anti-clockwise side of the reference line

hv
Flag determining the orientation of the annotation

= GXAXIS	Parallel to X axis (zero degrees)
= GYAXIS	Parallel to Y axis (90 degrees)

xory
Specified axis

= GXAXIS	X axis
= GYAXIS	Y axis

Description

The routine ggAddReferenceLine() draws a reference line across a graph at a specified value and labels it with a supplied text string.

The argument **xory** determines which axes the specified value **xyval** refers to and the reference line is drawn parallel to the opposite axes according to its current position and length. The supplied text string is then placed at one of the eight positions determined by the value of **labjus** and oriented according to the value of **hv**.

If the value **xyval** does not occur within the limits of the selected axes, neither the reference line or the text label is output. If the text position is out of range, a warning message is output and the default position is used.

See Also

Page 168

ggAddSquareWave

Syntax

```
C/C++:   void ggAddSquareWave(int npts, GPOINT *points, int pos, int xory);
```

```
F90:     subroutine ggAddSquareWave(npts, points, pos, xory)

         integer, intent(in) :: npts, pos, xory
         type(GPOINT), intent(in) :: points(*)
```

Arguments

npts
The number of points in the points array

points
Array of dimension **npts**, giving the X and Y axis values of the points that the square wave is fitted to

pos
Flag controlling the square wave interpolation. The value determines the position of the change in height from one point to the next

= GCURRENT	Change occurs at current point held in **points.x** and **points.y**
= GHALFWAY	Change occurs half way between current and next point
= GNEXT	Change occurs at next point

xory
The direction in which the square wave is fitted to the data set. The value of **xory** represents the axis to which the height change in the square wave is perpendicular to:

= GXAXIS	X axis
= GYAXIS	Y axis

Description

The routine ggAddSquareWave() takes the points held in the **points** array and fits a square wave to them. The square wave passes through all the given points. The changes in height can occur at one of three positions about the given points, depending on the value of **pos**.

See Also

Page 58

ggAddStepChartOutline

Syntax

C/C++:	**void ggAddStepChartOutline**(int nsteps, GSTEPCHART *steps, float base, int drop, int xory);

F90:	**subroutine ggAddStepChartOutline**(nsteps, steps, base, drop, xory) integer, intent(in) :: nsteps, drop, xory type(GSTEPCHART), intent(in) :: steps real, intent(in) :: base

Arguments

nsteps

The number of steps to be plotted

steps

Array of dimension **nsteps**, giving the start and finish width and height values of all the columns in the Step Chart

base

Base value in graphical coordinates which step widths are drawn to depending on the value of **drop**

drop

Flag determining how step edges are drawn

= GDROPTYPE0	Step heights only are drawn
= GDROPTYPE1	Link adjacent steps
= GDROPTYPE2	Link adjacent steps and draw non-adjacent edges to **base**
= GDROPTYPE3	Draw all step edges to **base**

xory

Flag determining which axis the data ranges are shown on, and on which axis the heights are shown

= GXAXIS	Data on X axis, heights on Y axis
= GYAXIS	Data on Y axis, heights on X axis

Description

The routine ggAddStepChartOutline() draws steps defined by the data ranges held in the components **steps.s** and **steps.f** and height values held in the component **steps.h**, with respect to either the current axes as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default axes used by one of the high level routines. The changes in step height are displayed in the style determined by the value of **drop**.

See Also

Page 101

ggAddStepChartValues

Syntax

C/C++:	**void ggAddStepChartValues**(int nsteps, GSTEPCHART *steps, float base, int sfl, int xory);

```
F90:    subroutine ggAddStepChartValues(nsteps, steps, base, sfl, xory)

        integer, intent(in) :: nsteps, sfl, xory
        type(GSTEPCHART), intent(in) :: steps
        real, intent(in) :: base
```

Arguments

nsteps

The number of steps to be annotated

steps

Array of dimension **nsteps**, giving the start and finish width and height values of all the columns in the Step Chart

base

The base value of all steps

sfl

Flag determining the value being output

= GSTART	Start of step (ie, **steps.s**)
= GFINISH	End of step (ie, **steps.f**)
= GWIDTH	Step width (ie, **steps.f** - **steps.s**)
= GHEIGHT	Step height (ie, **steps.h**)
= GHEIGHTABOVEBASE	Step height above **base** (ie, **steps.h** - **base**)

xory

Flag determining on which axis the values are plotted

= GXAXIS	X axis
= GYAXIS	Y axis

Description

The routine ggAddStepChartValues() displays the step, widths or heights, with respect to the current axes as set up by the last axis definition calls or through a previous call to ggPlotStepChart(). **xory** determines which direction the steps are displayed, thus if **xory**=GXAXIS the **steps.s** and **steps.f** values are displayed on the X axis and **steps.h** values are displayed on the Y axis. If ggAddStepChartValues() follows a call to ggPlotStepChart(), **xory** should be set to GXAXIS. Annotation of individual points about a Step Chart can be achieved using ggAddGraphValues().

The most recent call to ggSetValueAttribs() determines the position of the annotation about each area as well as the angle, justification and offset of the value string. The default is to position the required value at the centre of each area represented by the step, that is between **steps.s** and **steps.f** and between **steps.h** and the base. **steps.s**, **steps.f** and width values are output according to the format of the **xory** axis, where as height values are output according to the format of the other axis. The format is set by the most recent call to ggSetAxesAnnotation(). Prefix and suffix strings defined by the most recent call to ggSetValueTags() are appended to the numerical output.

If **sfl** is out or range, a warning message is output and the step heights values are displayed.

See Also

Page 103
ggAddGraphValues
ggPlotStepChart
ggSetValueAttribs
ggSetValueTags

ggAddVectors

Syntax

C/C++:

```
void ggAddVectors(int nx, int ny, GVECTOR *vectors, int head);
```

F90:

```
subroutine ggAddVectors(nx, ny, vectors, head)

type(GVECTOR), intent(in) :: vectors(nx, ny)
integer, intent(in) :: nx, ny, head
```

Arguments

nx
Number of vectors in the X direction

ny
Number of vectors in the Y direction

vectors
Two dimensional array of dimension **nx**,**ny** containing the directions, strengths and colours of each vector

head
Type of arrowhead drawn at each vector

= GCLOSED	Arrowhead closed (drawn as a triangle)
= GSOLID	Arrowhead filled using colour **vectors.col**
= GOPEN	Arrowhead open (drawn with two lines only)

Description

The routine ggAddVectors() draws a Vector Chart consisting of a grid of **nx** times **ny** arrows; each arrow having an individual strength, direction, and colour attribute determined by the corresponding values within the components **vectors.stren**, **vectors.direc**, and **vectors.col** respectively. The directions (in **vectors.direc**) are measured anticlockwise from the three o'clock (positive X axis) position.

No axes are drawn with this routine but the Vector Chart is mapped onto the area defined by the intersection of the current graphical axes as set up by the routines ggSetAxesPos() and ggSetAxesScaling() (or their defaults). The axes ranges and number of tick marks should be set appropriately for the Vector Chart data being displayed. The Vector Chart may be mapped onto a different area of the current axes system by using the routine ggSetVectorChartFrame().

The vectors are displayed as equally spaced arrows. The length of the arrows is linearly proportional to the strength values in **vectors.stren** and scaled using the current settings defined using ggSetVectorAttribs(). Arrows are not drawn if their strength is zero or if it lies outside the clipping limits defined by ggSetVectorLimits(). The arrows point in the corresponding directions held in the component **vectors.direc** (or that +180° if the strength is negative) and are displayed in the colour of the corresponding colour values held in **vectors.col**.

Three types of arrowhead, determined by the value of **head**, are available. If the type is out of range, a warning message is output and the default is used.

See Also

Page 123
ggSetAxesPos
ggSetAxesScaling
ggSetVectorChartFrame
ggSetVectorLimits
ggSetVectorAttribs

ggBlockFillAreaChart

Syntax

C/C++:	**void ggBlockFillAreaChart**(int nareas, GAREACHART *areas, int xory, int line);

```
F90:        subroutine ggBlockFillAreaChart(nareas, areas, xory, line)

            integer, intent(in) :: nareas, xory, line
            type(GAREACHART), intent(in) :: areas(*)
```

Arguments

nareas
The number of areas to be plotted

areas
Array of structures giving data for the Area Chart

xory
Flag determining which axis the data ranges are shown on, and on which axis the heights are shown

= GXAXIS	Data on X axis, heights on Y axis
= GYAXIS	Data on Y axis, heights on X axis

line
Line style to be used to fill each column

= GCURRENT	Specifies the current line style
= 1 -256	Specifies the line style index
>256	Specifies the current line style

Description The routine ggBlockFillAreaChart() draws block filled rectangles defined by the data ranges held in the components **areas.s** and **areas.f** and height values held in the components **areas.h1** and **areas.h2**, with respect to either the current axes, as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default axes used by one of the high level routines. The widths of the areas, held in **areas.s** and **areas.f**, are shown on the X axis if **xory**=GXAXIS, and on the Y axis if **xory**=GYAXIS.

All the areas are solid filled in the line style specified by **line** whereas the colour, depth and angle of the extrusions are determined by the current block fill settings as set by ggSetBlockChartAttribs(). The outline of each area is drawn in the current GINO line style.

See Also Page 107
ggPlotAreaChart
ggSetBlockChartAttribs

ggBlockFillBarChart

Syntax

```
C/C++:    void ggBlockFillBarChart(int nbars, GBARCHART *bars, float frac, int line);
```

```
F90:      subroutine ggBlockFillBarChart(nbars, bars, frac, line)

          type(GBARCHART), intent(in) :: bars(*)
          integer, intent(in) :: nbars, line
          real, intent(in) :: frac
```

Arguments

nbars
The number of bars in the Bar Chart

bars
Array of dimension **nbars**, giving the start and finish values of all the bars in the Bar Chart

frac
Fraction of an interval to be filled for each bar

line
Line style to be used to fill each column

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

Description

The routine ggBlockFillBarChart() block fills the bars of a Bar Chart, with respect to either the current axes, as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default axes used by ggPlotBarChart(). If a discrete axis has not been defined (using ggSetAxesScaling() with **scale**=GDISCRETE) or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

The area filled for each bar has the start and finish values defined in the components **bars.s** and **bars.f** and the width ((length of discrete axis)/**nbars**) * **frac**. If **frac** = 1.0, the fill occupies the whole interval.

The bars are centred on the tick marks on the discrete axis.

All the bars are solid filled in the line style specified by **line** whereas the colour, depth and angle of the extrusions are determined by the current block fill settings as set by ggSetBlockChartAttribs(). The outline of each area is drawn in the current GINO line style.

See Also

Page 95
ggPlotBarChart
ggSetBlockChartAttribs

ggBlockFillHistogram

Syntax

```
C/C++:     void ggBlockFillHistogram(int ncols, float *yarray, float frac, int line);
```

```
F90:       subroutine ggBlockFillHistogram(ncols, yarray, frac, line)

           integer, intent(in) :: ncols, line
           real, intent(in) :: yarray(*), frac
```

Arguments

yarray
Array, of dimension **ncols**, giving the heights of all the columns in the Histogram

ncols
The number of columns in the Histogram

frac
Fraction of an interval to be filled for each column

line
Line style to be used to fill each column

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

Description

The routine ggBlockFillHistogram() block fills the columns of a Histogram with respect to either the current axes, as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default Histogram axes used by ggPlotHistogram(). If a discrete axis has not been defined using ggSetAxesScaling(), or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

The area filled for the front of each column has the height defined in the array **yarray** and the width = ((length of X axis)/**ncols**) * **frac**. If **frac** = 1.0, the fill occupies the whole interval. The columns are centred on the tick marks on the discrete axis.

All the columns are solid filled in the line style specified by **line** whereas the colour, depth and angle of the extrusions are determined by the current block fill settings as set by ggSetBlockChartAttribs(). The outline of each area is drawn in the current GINO line style.

See Also

Page 88
ggPlotHistogram
ggSetBlockChartAttribs

ggBlockFillMultiHistogram

Syntax

C/C++:	**void ggBlockFillMultiHistogram**(int type, float *rdata, int ndim1, int ncols, int ndata,float frac, float gap, int line[], int is1, int is2);

F90:
```
subroutine ggBlockFillMultiHistogram(type, rdata, ndim1, ncols, ndata, frac, gap,
line, is1, is2)

integer, intent(in) :: type,ndim1,ncols,ndata
real, intent(in) :: rdata(ndim1,*), frac,gap
integer, intent(in) :: line(*),is1,is2
```

Arguments

type
Type of multi-histogram chart

= GSTACKED	Data sets stacked in single column
= GCLUSTERED	Data sets displayed as multiple columns

rdata
Two dimensional array giving the heights of the columns in each of the data sets

ndim1
The primary dimension of the data array **rdata**

ncols
The number of compound columns, or clusters of columns to be drawn. This can be from 1 to **ndim1**

ndata
The number of data sets for each column or column cluster to be drawn. This can be from 1 to the second dimension of the data array **rdata**

frac
Fraction of an interval to be filled for each column or column cluster

gap
Size of gap between members of the cluster, as a fraction of the width of a single member of the cluster. Range 0.0 to 1.0 (only used for type GCLUSTERED)

line
Integer array, of dimension **ndata**, determining the line style to be used to fill each data set. The corresponding component of each column will be drawn with the same line style

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

is1
The start position in the first dimension of the data array to be used as the first item (category) on the discrete axis

is2
The start position in the second dimension of the data array to be used as the first component on the continuous axis (or first column of cluster)

Description

The routine ggBlockFillMultiHistogram() block fills the columns of a multi-data set Histogram with respect to the current axes as set up by ggSetAxesPos() and ggSetAxesScaling(). If a discrete axis has not been defined using ggSetAxesScaling(), or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

The routine is designed to display a stacked or clustered histogram representing a block of data from an arbitrarily sized two dimensional array - **rdata**. Where **ndim1** is the primary dimension of the array and **is1** and **is2** specify the starting offset of the required data block. The dimensions of the data being represented is **ncols** by **ndata**, where **ncols** is the number of data items in each set and **ndata** is the number of data sets.

In C the array **rdata** should be declared: float rdata[][ndim1];

In Fortran 90 the array **rdata** should be declared: real rdata(ndim1,*)

The width of each stacked or clustered column = ((length of the discrete axis)/**ncols**) * **frac** where a **frac** = 0.99 will cause either the stacked or clustered column to nearly touch the adjacent column. Values of **frac** outside the range 0.0 to 1.0 are clipped to 0.1 and 0.9 respectively with values between 0.5 and 0.9 giving the most satisfactory results. Unlike the single data set histogram plot, the width of each column includes any extrusions due to the current block chart filling attributes, thus reducing the width of the front face of the column. This is required to ensure that each stacked or clustered column does not overlap with an adjacent column.

All the columns of each data set [i] are solid filled in the line style specified by **line[i]** whereas the colour, depth and angle of the extrusions are determined by the current block fill settings as set by ggSetBlockChartAttribs(). The outline of each column is drawn in the current GINO line style.

See Also

Page 113
ggBlockFillHistogram
ggSetBlockChartAttribs

ggBlockFillStepChart

Syntax

C/C++:	**void ggBlockFillStepChart**(int nsteps, GSTEPCHART *steps, float base, int xory, int line);

```
F90:    subroutine ggBlockFillStepChart(nsteps, steps, base, xory, line)
        integer, intent(in) :: nsteps, xory, line
        type(GSTEPCHART), intent(in) :: steps(*)
        real, intent(in) :: base
```

Arguments

nsteps
The number of steps to be plotted

steps
Array of dimension **nsteps**, giving the start and finish widths and height values of all the steps in the Step Chart

base
Base value in graphical coordinates which steps are filled to

xory
Flag determining which axis the data ranges are shown on, and on which axis the heights are shown

= GXAXIS	Data on X axis, heights on Y axis
= GYAXIS	Data on Y axis, heights on X axis

line
Line style to be used to fill each column

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

Description

The routine ggBlockFillStepChart() draws block filled rectangles defined by the data ranges held in the components **steps.s** and **steps.f** and between the **base** value on the axis on which the **steps.h** values are measured and the height values held in the component **steps.h**. The display of the rectangles is drawn with respect to either the current axes, as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default axes used by one of the high level routines. The widths of the columns, held in **steps.s** and **steps.f**, are shown on the X axis if **xory**=GXAXIS, and on the Y axis if **xory**=GYAXIS.

All the steps are solid filled in the line style specified by **line** whereas the colour, depth and angle of the extrusions are determined by the current block fill settings as set by ggSetBlockChartAttribs(). The outline of each area is drawn in the current GINO line style.

See Also

Page 101
ggSetBlockChartAttribs

ggConvertDates

Syntax

```
C/C++:      void ggConvertDates(int ndates, char *dates[ ], float data[ ]);
```

```
F90:        subroutine ggConvertDates(ndates, dates, data)

            integer, intent(in) :: ndates
            character*(*), intent(in) :: dates(*)
            real, intent(out) :: data(*)
```

Arguments

ndates
Number of dates to be converted

dates
Character array of dimension **ndates**, holding the dates in the current date input format

data
Returned date values for dates

Description Converts a character array of dates into an array of day numbers. These can then be used for plotting data in any of the graph display routines against an axis that has been set to have date scaling as set up by ggSetDateAxesScaling().

The dates are supplied as an array of character strings of up to 10 characters, formatted according to the current date input format type as set up by ggSetDateFormat(). Dates which are incorrectly formatted are returned as representing the date of January 1st 1900.

See Also Page 41
ggSetDateFormat
ggSetDateAxesScaling

ggConvertDateToGraph

Syntax

```
C/C++:      void ggConvertDateToGraph(char date[ ], float *value);
```

```
F90:        subroutine ggConvertDateToGraph(date, value)

            character*(*), intent(in) :: date
            real, intent(out) :: value
```

Arguments

date
Character string containing a single date in the current date input format

value
Returned day value

Description Converts a single date string into a real value representing the day number since September 13th 1752.

The date should be supplied as a string of up to 10 characters, formatted according to the current input date format type as set by ggSetDateFormat(). Dates which are incorrectly formatted are returned as representing the date of January 1st 1900.

See Also Page 41
ggSetDateFormat

ggConvertGraphToDate

Syntax

```
C/C++:      void ggConvertGraphToDate(float value, char date[ ]);
```

```
F90:        subroutine ggConvertGraphToDate(value, date)

            real, intent(in) :: value
            character*(*), intent(out) :: date
```

Arguments

value
Date value

date
Returned date string

Description

Converts a date data value into a character string representing its date in the current input date format.

The date is returned as a string of 10 characters, formatted according to the current input date type as set by ggSetDateFormat(). Such dates can be converted back to date values through the routine ggConvertDateToGraph().

See Also

Page 41
ggSetDateFormat
ggConvertDateToGraph

ggDefineMissingValues

Syntax

C/C++:	**void ggDefineMissingValues**(int mode, float val1, float val2, int xory);

```
F90:    subroutine ggDefineMissingValues(mode, val1, val2, xory)

        integer, intent(in) :: mode, xory
        real,    intent(in) :: val1, val2
```

Arguments

mode
Missing value mode

= GOFF,	Switch off missing values
= GEQUALTO,	Omit data equal to **val1** or equal to **val2**
= GGREATERTHAN,	Omit data greater than **val1**
= GGREATERTHANOREQUALTO,	Omit data greater than or equal to **val1**
= GLESSTHAN,	Omit data less than **val1**
= GLESSTHANOREQUALTO,	Omit data less than or equal to **val1**
= GOUTSIDERANGE,	Omit data outside the range **val1** to **val2**
= GINSIDERANGE,	Omit data equal to and inside the range **val1** to **val2**

val1
First value

val2
Second value

xory
Flag determining whether ggDefineMissingValues() refers to data on the X or the Y axis

= GXAXIS	X axis
= GYAXIS	Y axis

Description

The routine ggDefineMissingValues() defines one, two or a range of values that are omitted from some of the graph drawing routines. The values or range set in **val1**,**val2** may represent missing or rogue data within an application and are omitted from the graph according to the missing value **mode**. The specified values are checked against the graph data for either the X or the Y axis according to the **xory** setting.

ggDefineMissingValues() only affects polyline graphs, marker graphs and value graphs.

See Also

Page 69
ggPlotGraph
ggPlotXYPolarChart
ggAddGraphPolyline
ggAddGraphMarkers
ggAddGraphValues

ggDisplayFillColumn

Syntax

```
C/C++:    void ggDisplayFillColumn(float x, float y, int nfill, int *fill, int *line, char header[ ]);
```

```
F90:      subroutine ggDisplayFillColumn(x, y, nfill, fill, line, header)

          real, intent(in) :: x, y
          integer, intent(in) :: nfill, fill(*), line(*)
          character*(*), intent(in) :: header
```

Arguments

x
X coordinate of the top left hand corner of the column in the current units (default units are millimetres)

y
Y coordinate of the top left hand corner of the column in the current units

nfill
The number of filled rectangles

fill
Integer array, of dimension **nfill**, determining the fill styles to be used in each cell

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}

= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
> 256,	Specifies a solid fill for software fill, or the fill style index for hardware fill

line
Integer array, of dimension **nfill**, determining the line style to be used to fill the rectangle
The value of **line** is irrelevant where **fill** has a value less than -1

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

header
Column header.

Description

The routine ggDisplayFillColumn() outputs a set of rectangles in an optionally headed column with its top left corner positioned at **x**,**y** in user space coordinates. The column is divided into **nfill** or **nfill**+1 cells (depending on the header switch) into which the rectangles are drawn. A column frame is also drawn using the Text Chart frame line style index set by ggSetTextChartAttribs().

The rectangle in each cell is filled in the style determined by the combination of the corresponding elements of **fill** and **line**. Where a **fill** element is equal to GHOLLOW, only the boundary of the rectangle is drawn. One or more of the cells may be left unfilled by giving the corresponding element(s) of **fill** a value of -2 or less. Negative values of **line** cause a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

The string **header** is output in an additional header cell if headers are switched on in the previous call to ggSetTextChartAttribs(). If displayed, the string is output centrally in the cell using the current GINO text and line attributes.

See Also

Page 160
ggSetTextChartAttribs

ggDisplayGeneratedColumn

Syntax

C/C++:
```
void ggDisplayGeneratedColumn(float x, float y, int nval, float vbeg, float vend, char header[ ]);
```

F90:
```
subroutine ggDisplayGeneratedColumn(x, y, nval, vbeg, vend, header)

real, intent(in) :: x, y, vbeg, vend
integer, intent(in) :: nval
character*(*), intent(in) :: header
```

Arguments

x
X coordinate of the top left hand corner of the column in the current units (default units are millimetres)

y
Y coordinate of the top left hand corner of the column in the current units

nval
The number of values to be output between and including **vbeg** and **vend**

vbeg
First value to be output

vend
Last value to be output

header
Column header

Description

The routine ggDisplayGeneratedColumn() generates a set of values and outputs them in an optionally headed column with its top left corner positioned at **x**,**y** in user space coordinates. The column is divided into **nval** or **nval**+1 cells (depending on the header switch) into which the generated values are placed according to the current Text Chart justification as set by ggSetTextChartAttribs(). A column frame is also drawn using the Text Chart frame colour index also set by ggSetTextChartAttribs().

nval numbers are generated equally spaced from **vbeg** to **vend** inclusively. For example, if **vbeg** = 1, **vend** = 7 and **nval** = 4 then the numbers 1, 3, 5, and 7 are output. The values are output, using the current GINO text and line attributes, in the format of the current Y axis annotation as set by ggSetAxesAnnotation(). Prefix and/or suffix strings may be added to each value in the column using the routine ggSetValueTags().

The string **header** is output in an additional header cell if headers are switched on in the previous call to ggSetTextChartAttribs(). If displayed, the string is output centrally in the cell using the current GINO text and line attributes.

See Also

Page 157
ggSetAxesAnnotation
ggSetTextChartAttribs
ggSetValueTags

ggDisplayLineColumn

Syntax

C/C++:	**void ggDisplayLineColumn**(float x, float y, int nline, int *line, int ang, char header[ ]);

F90:

```
subroutine ggDisplayLineColumn(x, y, nline, line, ang, header)

real, intent(in) :: x, y
integer, intent(in) :: nline, line(*), ang
character*(*), intent(in) :: header
```

Arguments

x
X coordinate of the top left hand corner of the column in the current units (default units are millimetres)

y
Y coordinate of the top left hand corner of the column in the current units

nline
The number of lines

line
Integer array, of dimension **nline**, determining the line style to be used in each cell

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

ang
Flag determining which angle the lines are displayed

= GHORIZONTAL	Horizontal
= GVERTICAL	Vertical
= GRIGHTDIAGONAL	Bottom left to top right
= GLEFTDIAGONAL	Top left to bottom right

header
Column header

Description

The routine ggDisplayLineColumn() outputs a set of lines in an optionally headed column with its top left corner positioned at **x**,**y** in user space coordinates. The column is divided into **nline** or **nline**+1 cells (depending on the header switch) into which the lines are drawn. A column frame is also drawn using the Text Chart frame line style index set by ggSetTextChartAttribs().

The line in each cell is drawn in the style of the corresponding index held in the array **line**, each element pointing to an entry in the GINO line style table. If the array **line** contains any negative line style indices, a warning message is output and the absolute value is used. The line is drawn in one of four directions depending on the value of **ang**. If the value of **ang** is out of range, option 3 is used.

The string **header** is output in an additional header cell if headers are switched on in the previous call to ggSetTextChartAttribs(). If displayed, the string is output centrally in the cell using the current GINO text and line attributes.

See Also

Page 160
ggSetTextChartAttribs

ggDisplayMarkerColumn

Syntax

C/C++:	**void ggDisplayMarkerColumn**(float x, float y, int nsym, int *sym, int *line, char header[ ]);

```
F90:    subroutine ggDisplayMarkerColumn(x, y, nsym, sym, line, header)

        real, intent(in) :: x, y
        integer, intent(in) :: nsym, sym(*), line(*)
        character*(*), intent(in) :: header
```

Arguments

x
X coordinate of the top left hand corner of the column in the current units (default units are millimetres)

y
Y coordinate of the top left hand corner of the column in the current units

nsym
Number of symbols in the column

sym
Array of length **nsym** containing symbols to be output in each cell

= GSPOT	Dot
= GUP	Standard GINO symbol
= GDOWN	Standard GINO symbol
= GPLUS	Standard GINO symbol
= GCROSS	Standard GINO symbol
= GBOX	Standard GINO symbol
= GDIAMOND	Standard GINO symbol
= GCIRCLE	Standard GINO symbol
= GSTAR	Standard GINO symbol
= 9 to 23	Optional hardware symbol
> 23	Character from font table

line
Array of dimension **nsym**, determining the line style to be used for the corresponding symbol

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

header
Column header

Description

The routine ggDisplayMarkerColumn() outputs a set of symbols in an optionally headed column with its top left corner positioned at **x,y**. The column is divided into **nsym** or **nsym**+1 cells (depending on the header switch) into which the symbols are drawn. A column frame is also drawn using the Text Chart frame line style index set by ggSetTextChartAttribs().

The symbol numbers required for each cell are held in the array **sym**. Any of the valid GINO standard or font symbol numbers may be used. Negative symbol numbers or symbols that are not available default to symbol GUP. They are drawn centrally within each cell in the line style of the corresponding index held in the array **line**, each element pointing to an entry in the GINO line style table. If the array **line** contains any negative line style indices, a warning message is output and the absolute value is used.

The string **header** is output in an additional header cell if header cells have been switched on in the previous call to ggSetTextChartAttribs(). If displayed, the string is output centrally in the cell using the current GINO attributes.

See Also

Page 160
ggSetTextChartAttribs

ggDisplayPercentageColumn

Syntax

```
C/C++:    void ggDisplayPercentageColumn(float x, float y, int nval, float *values, char header[
          ]);
```

```
F90:      subroutine ggDisplayPercentageColumn(x, y, nval, values, header)

          real, intent(in) :: x, y, values(*)
          integer, intent(in) :: nval
          character*(*), intent(in) :: header
```

Arguments

x
X coordinate of the top left hand corner of the column in the current units (default units are millimetres)

y
Y coordinate of the top left hand corner of the column in the current units

nval
The number of values to be output in the column

values
Array of length **nval** containing the original values from which the percentages are calculated

header
Column header

Description

The routine ggDisplayPercentageColumn() generates a set of percentage values and outputs them in an optionally headed column with its top left corner positioned at **x**,**y** in user space coordinates. The column is divided into **nval** or **nval**+1 cells (depending on the header switch) into which the generated values are placed according to the current Text Chart justification as set by ggSetTextChartAttribs(). A column frame is also drawn using the Text Chart frame colour index also set by ggSetTextChartAttribs().

nval numbers are generated as percentages of the total of all the values held in the array **values**. The values are output, using the current GINO text and line attributes, in the format of the current Y axis annotation as set by ggSetAxesAnnotation(). The values are followed by a '%' sign. If the total is zero, an error message is output and no values are displayed.

The string **header** is output in an additional header cell if headers are switched on in the previous call to ggSetTextChartAttribs(). If displayed, the string is output centrally in the cell using the current GINO text and line attributes.

See Also

Page 157
ggSetAxesAnnotation
ggSetTextChartAttribs

ggDisplayStringColumn

Syntax

```
C/C++:    void ggDisplayStringColumn(float x, float y, int nstr, char *string[ ], char header[ ]);
```

```
F90:      subroutine ggDisplayStringColumn(x, y, nstr, string, header)

          real, intent(in) :: x, y
          integer, intent(in) :: nstr
          character*(*), intent(in) :: string(*), header
```

Arguments

x
X coordinate of the top left hand corner of the column in the current units (default units are millimetres)

y
Y coordinate of the top left hand corner of the column in the current units

nstr
The number of strings in the column

string
Array of character strings to be output

header
Column header

Description

The routine ggDisplayStringColumn() output a set of character strings in an optionally headed column with its top left corner positioned at **x**,**y** in user space coordinates. The column is divided into **nstr** or **nstr**+1 cells (depending on the header switch) into which the strings are placed according to the current Text Chart justification as set by ggSetTextChartAttribs(). A column frame is also drawn using the Text Chart frame colour index also set by ggSetTextChartAttribs().

The string **header** is output in an additional header cell if headers are switched on in the previous call to ggSetTextChartAttribs(). If displayed, the string is output centrally in the cell using the current GINO text and line attributes.

See Also

Page 155
ggSetTextChartAttribs

ggDisplayValueColumn

Syntax

C/C++:	**void ggDisplayValueColumn**(float x, float y, int nval, float *values, char header[ ]);

```
F90:    subroutine ggDisplayValueColumn(x, y, nval, values, header)

        real, intent(in) :: x, y, values(*)
        integer, intent(in) :: nval
        character*(*), intent(in) :: header
```

Arguments

x
X coordinate of the top left hand corner of the column in the current units (default units are millimetres)

y
Y coordinate of the top left hand corner of the column in the current units

nval
The number of values to be output in the column

values
Array of length **nval** containing the values to be output

header
Column header

Description

The routine ggDisplayValueColumn() outputs a set of values in an optionally headed column with its top left corner positioned at **x**,**y** in user space coordinates. The column is divided into **nval** or **nval**+1 cells (depending on the header switch) into which the values are placed according to the current Text Chart justification as set by ggSetTextChartAttribs(). A column frame is also drawn using the Text Chart frame colour index also set by ggSetTextChartAttribs().

The values are held in the array **values** and are output, using the current GINO text and line attributes, in the format of the current Y axis annotation as set by ggSetAxesAnnotation(). Prefix and/or suffix strings may be added to each value in the column using the routine ggSetValueTags().

The string **header** is output in an additional header cell if headers are switched on in the previous call to ggSetTextChartAttribs(). If displayed, the string is output centrally in the cell using the current GINO text and line attributes.

See Also

Page 157
ggSetAxesAnnotation
ggSetTextChartAttribs
ggSetValueTags

ggDrawArrow

Synopsis :

C/C++:	**void ggDrawArrow**(float xhead, float yhead, int head, int mode);

F90:	**subroutine ggDrawArrow**(xhead, yhead, head, mode) real, intent(in) :: xhead, yhead integer, intent(in) :: head, mode

Arguments

xhead

Value that specifies the X part of the user space coordinates (if **mode** = GSPACE) or the X part of the graphical axes coordinates (if **mode** = GGRAPH), to which the line of the arrow is to be drawn, and at which point the arrowhead is to be drawn

yhead

Value that specifies the Y part of the user space coordinates (if **mode** = GSPACE) or the Y part of the graphical axes coordinates (if **mode** = GGRAPH), to which the line of the arrow is to be drawn, and at which point the arrowhead is to be drawn

head

The type of arrowhead to be drawn

= GOPEN	Arrowhead open (drawn with two lines only)
= GCLOSED	Arrowhead closed (drawn as a triangle)
= GSOLID	Arrowhead filled using solid colour in the current line style

mode

Flag specifying whether the point (**xhead**,**yhead**) represents user space coordinates or graphical axes coordinates

= GSPACE	(**xhead**,**yhead**) gives the space coordinates
= GGRAPH	(**xhead**,**yhead**) gives the graphical axes coordinates

Description

The routine ggDrawArrow() draws a line from the current pen position to the point (**xhead**, **yhead**), and draws an arrowhead at that point. The arrowhead is made up of two or three (depending on **head**) sides of an equilateral triangle. The length of the sides of the triangle is the current character width, unless the tail of the arrow is less than one character width long, in which case the length of the sides of the triangle is half the length of the tail of the arrow. If **head** is not GOPEN, GSOLID or GCLOSED, a default of GOPEN is assumed. If **mode** is not GSPACE or GGRAPH, a default of GSPACE is assumed.

See Also

Page 168
ggMoveToGraphPoint
ggAddGraphLine

ggDrawAxes

Syntax

C/C++:	**void ggDrawAxes**(int tick, int tickside, int val, int xory);

F90:	**subroutine ggDrawAxes**(tick, tickside, val, xory) integer, intent(in) :: tick, tickside, val, xory

Arguments

tick

The style in which tick marks are to be drawn on the axis

= GNONE	No tick marks drawn
= GCARDINAL	Tick marks drawn only at the defined intervals
= GINTERMEDIATE	Tick marks drawn at the defined intervals and also at intermediate positions

tickside

The side of axes tickmarks are drawn

= GCLOCKWISE	Tick marks drawn on the clockwise side of the axis
= GANTICLOCKWISE	Tick marks drawn on the anticlockwise side of the axis

val

The way in which numeric annotation is to be written on the axis

= GNOVAL	No numeric annotation written
= GCLOCKWISE	Numeric annotation written on clockwise side of axis
= GANTICLOCKWISE	Numeric annotation written on the anti-clockwise side of the axis

xory

Specified axis

= GXAXIS	The X axis is drawn
= GYAXIS	The Y axis is drawn

Description

The routine ggDrawAxes() draws the **xory** axis defined by ggSetAxesPos() and/or ggSetAxesScaling(), or the default axis if these routines have not been called. For the default values, see ggSetAxesPos() and ggSetAxesScaling().

By default, on a linearly-scaled axis, numeric scale values are written in the form:
$S = P * 10^N$

where

S is the true value of an axis tick mark

P is the actual number written by the axis tick mark with up to two decimal places

$* 10^N$ is a multiplier written at the end of the axis ensuring N is not in the range -2 to 2

This may be altered using the routine ggSetAxesAnnotation().

On a logarithmically-scaled axis, the numeric annotation is a set of consecutive integers, N representing $\log_{10}(10^N)$. Intermediate tick marks may be drawn depending on the available space. The scale factor LOG10 is written at the end of the axis.

A discrete axis is similar to a linear axis, except that the tick marks and numeric annotation are centred in the intervals, and no intermediate tick marks are drawn.

General axis annotation attributes, such as positioning, start and skip values, angled, justified, and reduced text, may be set using the routine ggSetAxesAttribs().

See Also

Page 24
ggSetAxesAnnotation
ggSetAxesAttribs

ggDrawAxesLabels

Syntax

```
C/C++:     void ggDrawAxesLabels(int nstr, char *string[ ], int val, int xory);
```

```
F90:       subroutine ggDrawAxesLabels(nstr, string, val, xory)

           integer, intent(in) :: nstr,val,xory
           character*(*), intent(in) :: string
```

Arguments

string
Character array of dimension **nstr**, holding the labels

nstr
The number of labels to be written

val
Flag indicating on which side of the axis the annotation should be shown

= GNOVAL	No annotation
= GCLOCKWISE	Clockwise of axis
= GANTICLOCKWISE	Anticlockwise of axis

xory
Flag determining the axis to be labelled and label positioning

= GXAXIS	Annotation along the current X axis
= GYAXIS	Annotation along the current Y axis

Description

The routine ggDrawAxesLabels() displays the labels in **string** along the axis denoted by **xory**.

If the number of tick marks on an axis is greater than the number of labels in **string**, the labels are repeated. By default the labels are centred on the tick marks, and are only included if there is sufficient space. If the interval is too small for the labels, they are written at alternate tick marks or every third tick mark etc. This default output form can be changed with the routine ggSetAxesAttribs() which gives control over the position, angle, justification and other attributes of the annotation.

Axes drawn with ggDrawAxes() or ggAddGrid() should have numeric output suppressed if textual labels are required at the same position.

See Also

Page 30
ggSetAxesAttribs
ggDrawAxes
ggSetAxesPos
ggSetAxesScaling
ggAddGrid

ggDrawAxesTitle

Syntax

C/C++:	**void ggDrawAxesTitle**(char string[ ], float yorx, int xory, int pos1, int pos2);

```
F90:          subroutine ggDrawAxesTitle(string, yorx, xory, pos1, pos2)

              character*(*), intent(in) :: string
              real, intent(in) :: yorx
              integer, intent(in) :: xory, pos1, pos2
```

Arguments

string
Variable or constant holding the title

yorx
Value representing the Y or X user space coordinate positioning the title text

xory
Flag determining the axis to be titled

= GXAXIS	Title the current X axis
= GYAXIS	Title the current Y axis

pos1
Justification of string relative to **yorx**

= GBOTTOM	Title written with the bottom of the characters at **yorx**
= GTOP	Title written with the top of the characters at **yorx**

pos2
The position of the title relative to the axis

= GBOTTOM	Title justified to the bottom of the axis
= GLEFT	Title justified to the left of the axis
= GCENTRE	Title justified to the centre of the axis
= GRIGHT	Title justified to the right of the axis
= GTOP	Title justified to the top of the axis

Description

The routine ggDrawAxesTitle() outputs a title held in **string** with reference to either the X or Y axis.

yorx is the Y or X coordinate positioning the title. If the X axis is being titled, the Y coordinate of the text is constant and contained in **yorx**, and the X coordinate is determined by the flags **pos1** and **pos2**. If the Y axis is being titled, **yorx** contains the X coordinate of the text and **pos1**,**pos2** determines the Y coordinate.

xory determines the axis being titled, and the positioning of the title. For the X axis the position determined by **yorx** can specify the top of the characters in the title or the bottom of the characters. For the Y axis, the characters are rotated through 90 degrees so that the top of each character is the leftmost point of the character as drawn, and the bottom of each character is the rightmost point of the character as drawn.

pos2 determines the justification of the title relative to the axis. X axis titles are written to be read from left to right, and Y axis titles are written with the first character lowest.

The title need not be written along the axis itself. For example, a title can be written at the top of a graph by using the routine with **pos1 =** GXAXIS and **yorx** set to a Y coordinate above the top of the graph. ggDrawGraphTitle() provides additional graph titling facilities.

See Also

Page 29
ggDrawGraphTitle

ggDrawGraphTitle

Syntax

C/C++:	**void ggDrawGraphTitle**(char string[ ], int xpos, int ypos);

```
F90:      subroutine ggDrawGraphTitle(string, xpos, ypos)

          integer, intent(in) :: xpos, ypos
          character*(*), intent(in) :: string
```

Arguments

string
Variable or constant holding the graph title

xpos
Flag determining the X position of the title within the current graph drawing limits

= GLEFT	Left justified
= GCENTRE	Centre justified
= GRIGHT	Right justified

ypos
Flag determining the Y position of the title within the current graph drawing limits

= GTOP	Top
= GMIDDLE	Middle
= GBOTTOM	Bottom

Description

The routine ggDrawGraphTitle() displays a text string as a title in one of nine positions within the current graph drawing limits. The position is determined by the combination of the values **xpos** and **ypos**.

The graph drawing limits are set using the routine ggSetPlotFrame() or are defined as being the current GINO window limits if this routine has not been called. The title is output using the current GINO character and font attributes (except angle and justification).

Any trailing spaces in **string** are ignored on output and a maximum of 140 characters can be used in the graph title.

See Also

Page 165
ggSetPlotFrame

ggDrawPolarAxes

Syntax

```
C/C++:    void ggDrawPolarAxes(int tick1, int tick2, int val, int nintp, float vendp, int rorth);
```

```
F90:      subroutine ggDrawPolarAxes(tick1, tick2, val, nintp, vendp, rorth)

          integer, intent(in) :: tick1, tick2, val, nintp, rorth
          real, intent(in) :: vendp
```

Arguments

tick1
Flag determining the way in which tick marks are to be drawn on the axis.

= GNONE	No tick marks drawn (for THETA axis, the axis is also omitted)
= GCARDINAL	Tick marks drawn at the defined **nintp** intervals
= GINTERMEDIATE	Tick marks drawn at the defined **nintp** intervals and at intermediate positions

tick2
The way in which tickmarks are to be drawn

= GTICKS	Tick marks only
= GTICKSANDRADII	Tick marks and radii (THETA axis)
= GTICKSANDCIRCLES	Tick marks and circles (R axis)

val
Flag determining whether annotation is drawn

= GNOANNOTATION	No annotation
= GANNATATION	Annotation drawn

nintp
The number of intervals for the tick marks. If **tick2** = GTICKSANDRADII or GTICKSANDCIRCLES, **nintp** specifies the number of circles (R axis) or radii (THETA axis).

vendp
Value specifying the end of the range to be included on the axis; **vendp** is specified in current units for the R axis radius and in degrees for the THETA axis.

rorth
Number whether ggDrawPolarAxes() refers to the R or THETA axis.

= GRAXIS	The R axis is drawn
= GTHETAAXIS	The THETA axis is drawn

Description

The routine ggDrawPolarAxes() draws the **rorth** polar axis defined by ggSetPolarChartAttribs() or the default axis if this routine has not been called.

The numeric annotation is displayed on the zero THETA axis in the same form as normal axes and around the maximum R limit in integer degrees. ggDrawPolarAxes() alters the settings of ggSetAxesPos() and ggSetAxesScaling() and therefore these should be reset if a normal graph is required after ggDrawPolarAxes() is used.

Axis annotation attributes, such as format, positioning, start and skip values, angled, justified, and reduced text, may be set using the routines ggSetAxesAnnotation() and/or ggSetAxesAttribs().

When selecting tick marks and radii/circles, **tick2** may be set to GTICKSANDRADII or GTICKSANDCIRCLES as these two options are equivalent to each other. Whether ticks are drawn with radii or circles is decided by the the value of **rorth**.

See Also

Page 133
ggSetAxesAnnotation
ggSetAxesAttribs
ggSetPolarChartAttribs

ggEnqAxesAnnotation

Syntax

```
C/C++:    void ggEnqAxesAnnotation(int *ndp, int *npower, int *nrfigs, int *asty, int xory);
```

```
F90:      subroutine ggEnqAxesAnnotation(ndp, npower, nrfigs, asty, xory)

          integer, intent(out) :: ndp, npower, nrfigs, asty
          integer, intent(in) :: xory
```

Arguments

ndp
The current number of decimal places for graph axes annotation. **ndp** is in the range -9 to 9

npower
The current power to which the annotation is forced. **npower** is in the range -15 to 15

nrfigs
The current field width for graph axes annotation

asty
The current annotation scale type of display

= GNOSCALE	No scale factor
= GSCALEPOWEROF10	Scale factor is displayed as $*10^n$
= GSCALEZEROS	Scale factor is displayed as ‘000 or 0.0’
= GSCALEWORD	Scale factor is displayed in words
= GSCALEPREFIX	Scale factor is displayed in engineering units

xory
Flag determining which annotation characteristics are returned

= GXAXIS	Parameters returned for X axis
= GYAXIS	Parameters returned for Y axis

Description The routine ggEnqAxesAnnotation() returns the current numerical annotation settings as defined by the most recent call to ggSetAxesAnnotation(). The values returned refer to the settings for a particular axis defined by **xory**.

See Also Page 29
ggSetAxesAnnotation

ggEnqAxesAttribs

Syntax

C/C++:
```
void ggEnqAxesAttribs(int *switch, float *xy, int *nstrt, int *nsk, float *aoffs, float
*angstr, int *jstmb, int *jslcr, int *reduc, int xory);
```

F90:
```
subroutine ggEnqAxesAttribs(switch, xy, nstrt, nsk, aoffs, angstr, jstmb, jslcr, reduc,
xory)

integer, intent(out) :: switch, nstrt, nsk, jstmb, jsclr, reduc
real, intent(out) :: xy, aoffs, angstr
integer, intent(in) :: xory
```

Arguments

switch
Annotation position

= GONAXIS	specified axis
= GOFFSET	Positioned at **xory**

xy
The position in user coordinates of the annotation in either the X or Y direction. **xory** is used depending on the value of **switch**

nstrt
Tick mark number at which the annotation starts

nsk
Number of annotation elements to be skipped during annotation

= -1	Automatic skip generation
= 0	No labels skipped
> 0	Skip **nsk** elements

aoffs
Offset as a proportion of distance between major tick marks on the specified axis

angstr
Annotation string angle

jstmb
Vertical justification for each annotation element

= GTOP	Top justified - string below control point
= GMIDDLE	Middle justified - string centred at control point
= GBOTTOM	Bottom justified - string above control point
= GDEFAULTPOSITION	Default for requested axis

jslcr
Horizontal justification for each annotation element

= GLEFT	Left justified
= GCENTRE	Centre justified
= GRIGHT	Right justified
= GDEFAULTPOSITION	Default for requested axis

reduc
Annotation character size reduction switch

= GNOREDUCE	No size reduction
= GREDUCE	Reduced annotation elements to avoid overlapping

xory
Specified axis

= GXAXIS	Parameters set for X axis
= GYAXIS	Parameters set for Y axis

Description

The routine ggEnqAxesAttribs() returns the current axes annotation settings as defined by the most recent call to ggSetAxesAttribs(). The values returned refer to the settings for a particular axis defined by **xory**.

See Also

Page 38
ggSetAxesAttribs

ggEnqAxesPos

Syntax

```
C/C++:   void ggEnqAxesPos(int *or, GPOINT *origin, float *axlen, int xory);
```

```
F90:     subroutine ggEnqAxesPos(or, origin, axlen, xory)

         integer, intent(out) :: or
         type (GPOINT), intent(out) :: origin
         real, intent(out) :: axlen
         integer, intent(in) :: xory
```

Arguments

or
An integer determining the position of the point (**origin**) on the axis

= GDATAORIGIN	(**origin**) represents the point at which the natural origin should occur on the drawing area

= GAXISSTART	(**origin**) represents the point where the axis starts on the drawing area

origin.x
Value which specifies the X part of the user space coordinates defining the position of the axis in the drawing area

origin.y
Value which specifies the Y part of the user space coordinates defining the position of the axis in the drawing area

axlen
Value specifying the length of the axis in the current units (default units are millimetres)

xory
Flag determining which position characteristics are returned

= GXAXIS	Parameters returned for X axis
= GYAXIS	Parameters returned for Y axis

Description The routine ggEnqAxesPos() returns the position and length of either axis as defined by the most recent call to ggSetAxesPos() (or the default setting).

See Also Page 23
ggSetAxesPos

ggEnqAxesScaling

Syntax

```
C/C++:    void ggEnqAxesScaling(int *scale, int *nints, float *vbeg, float *vend, int xory);
```

```
F90:      subroutine ggEnqAxesScaling(scale, nints, vbeg, vend, xory)

          integer, intent(out) :: scale, nints
          real, intent(out) :: vbeg, vend
          integer, intent(in) :: xory
```

Arguments

scale
The current type of scaling for requested axis. Type of scaling is as set by previous calls to ggSetAxesScaling() or ggSetPolarChartAttribs()

nints
The current number of intervals on the requested axis. For scale type GLINEARTYPE1, this may not be the same as set in ggSetAxesScaling() or ggDrawPolarAxes(). For log scaling, **nints** is returned as zero

vbeg
Real value giving the initial data value of the range on the axis requested. The value may be rounded down from that specified by the previous call to ggSetAxesScaling() or ggDrawPolarAxes()

vend
Real number specifying the end of the range on the axis requested. The value may be rounded up from that specified by the previous call to ggSetAxesScaling() or ggDrawPolarAxes()

xory
Flag determining which scaling characteristics are returned

= GXAXIS	Parameters returned for X axis
= GYAXIS	Parameters returned for Y axis

Description

The routine ggEnqAxesScaling() returns the scaling parameters of either the X or Y axis as they will be or have been displayed by the axes drawing routines (ggDrawAxes() or ggDrawPolarAxes()). The returned arguments may not be exactly as specified, as for certain scale types the number of intervals and the range is adjusted to give more sensible output.

See Also

Page 23
ggSetAxesScaling
ggSetPolarChartAttribs
ggDrawPolarAxes

ggEnqBlockChartAttribs

Syntax

```
C/C++:    void ggEnqBlockChartAttribs(int *coloff, float *azim, float *elev, float *depth, float
          *top, float *side);
```

```
F90:      subroutine ggEnqBlockChartAttribs(coloff, azim, elev, depth, top, side)

          integer, intent(out) :: coloff
          real, intent(out) :: azim, elev, depth, top, side
```

Arguments

coloff
Colour index offset for shading (default =20)

azim
Azimuth angle of block in range -60.0 to 60.0 (default =30.0)

elev
Elevation angle of block in range -60.0 to 60.0 (default= = 30.0)

depth
Depth as fraction of column width in range 0.1 to 10.0 (default = 1.0)

top
The relative lightness of the top of a column compared to that of the front in range 0.0 to 100.0 (default =0.67)

side
The relative lightness of the side of a column compared to that of the front in range 0.0 to 100.0 (default = 0.33)

Description

The routine ggEnqBlockChartAttribs() enquires the Block Chart attributes used for all the Block Chart routines. A full description of the attributes is documented with the setting routine ggSetBlockChartAttribs(). The default block chart attributes may be restored using the routine ggRestoreBlockChartAttribs().

See Also

Page 87
ggBlockFillAreaChart
ggBlockFillBarChart
ggBlockFillHistogram
ggBlockFillMultiHistogram
ggBlockFillStepChart
ggRestoreBlockChartAttribs
ggSetBlockChartAttribs

ggEnqDateAxesAnnotation

Syntax

```
C/C++:   void ggEnqDateAxesAnnotation(int *fdow, int *fday, int *fmon, int *fyear, int xory);
```

```
F90:     subroutine ggEnqDateAxesAnnotation(fdow, fday, fmon, fyear, xory)

         integer, intent(out) :: fdow, fday, fmon, fyear
         integer, intent(in) :: xory
```

Arguments

fdow
Format for Day of Week

< 0	Alphanumeric with -**fdow** characters (up to 9)
= GNONE	Not present
= 1	Numeric (1 to 7) with 1 = Monday

fday
Format for Day

< 0	Alphanumeric (1st, 2nd, 3rd, etc.)
= GNONE	Not present
= 1	Numeric (1 to 31)

fmon
Format for Month

< 0	Alphanumeric with -**fmon** characters (up to 9)
= GNONE	Not present
= 1	Numeric (1 to 12)

fyear
Format for Year

= GNONE	Not present
= 2	Two digit display
= 4	Four digit display

xory
Flag determining which date annotation characteristics are returned

= GXAXIS	The X axis setting is returned
= GYAXIS	The Y axis setting is returned

Description

Returns the output format for date axis annotation as set by ggSetDateAxesAnnotation(). Date axis scaling is defined using ggSetDateAxesScaling().

See Also

Page 45
ggSetDateAxesAnnotation
ggSetDateAxesScaling

ggEnqDateAxesScaling

Syntax

C/C++:	**void ggEnqDateAxesScaling**(int *scale, int *dincr, char dbeg[ ], char dend[ ], int xory);

```
F90:   subroutine ggEnqDateAxesScaling(scale, dincr, dbeg, dend, xory)

       integer, intent(out) :: scale, dincr
       character*(*), intent(out) :: dbeg, dend
       integer, intent(in) :: xory
```

Arguments

scale
Axis scaling type

< 0	Current axis is numerically scaled
= GLINEARTYPE1	}
= GLINEARTYPE2	} Date axis scaling type
= GLINEARTYPE3	}

dincr
The increment being used for major tick marks on the date axis

= GDECADE	Decade (10 years)
= GYEAR	Year
= GMONTH	Month
= GWEEK	Week (7 days starting on Monday)
= GDAY	Day

dbeg
Date at the beginning of the date axis

dend
Date at the end of the date axis

xory
Flag determining which date settings are returned

= GXAXIS	The X axis setting is returned
= GYAXIS	The Y axis setting is returned

Description

The routine ggEnqDateAxesScaling() returns the current date axis scaling information for either the X or Y axis as set up by ggSetDateAxesScaling().

The values returned are not necessarily those requested by ggSetDateAxesScaling() as that routine may select an appropriate date interval or adjust the start and end dates on the axis according to the requested scaling type. The values returned are therefore those that represent the actual axis to be, or already drawn.

The start and end dates are returned as character strings of 10 characters, in the current input date format as set by ggSetDateFormat().

This routine should only be called in association with a corresponding call to ggSetDateAxesScaling() for the same axis. If the requested axis scaling has been defined through a call to ggSetAxesScaling() (ie. numeric scaling). **scale** will return with a negative value and other values are meaningless.

See Also

Page 45
ggSetAxesScaling
ggSetDateFormat
ggSetDateAxesScaling

ggEnqDateFormat

Syntax

```
C/C++:    void ggEnqDateFormat(int *inform, char *insep, int *ouform, char *ousep);
```

```
F90:      subroutine ggEnqDateFormat(inform, insep, ouform, ousep)

          integer, intent(out) :: inform, ouform
          character, intent(out) :: insep, ousep
```

Arguments

inform
Date format for input dates

= GBRITISH	British form dd/mm/yy or dd/mm/yyyy
= GAMERICAN	American form mm/dd/yy or mm/dd/yyyy
= GLOGICAL	Standard Logical form yy/mm/dd or yyyy/mm/dd

insep
Date separator for input dates

ouform
Date format for output dates

= GBRITISH	British form dd/mm/yy or dd/mm/yyyy
= GAMERICAN	American form mm/dd/yy or mm/dd/yyyy
= GLOGICAL	Standard Logical form yy/mm/dd or yyyy/mm/dd

ousep
Date separator for output dates

Description

Returns the current date format for numeric date character strings as set by ggSetDateFormat(). The input format is used by ggConvertDateToGraph(), ggConvertDates(), ggSetDateAxesScaling(), ggEnqDateAxesScaling() and ggConvertGraphToDate() routines.

See Also

Page 45
ggConvertDates
ggSetDateFormat
ggSetDateAxesScaling

ggEnqGridMarker

Syntax

C/C++:	**void ggEnqGridMarker**(int *sym);

F90:	**subroutine ggEnqGridMarker**(sym) integer, intent(out) :: sym

Arguments

sym
Grid intersection symbol number

Description

The routine ggEnqGridMarker() returns the grid intersection symbol as defined by ggSetGridMarker(). The default being a cross symbol (no. 3, GPLUS) displayed at the intersection of the major tick marks.

See Also

Page 27
ggSetGridMarker

ggEnqPieChartAnnotation

Syntax

C/C++:	**void ggEnqPieChartAnnotation**(int *type, int *txt, int *per, int *val, float *tol);

F90:	**subroutine ggEnqPieChartAnnotation**(type, txt, per, val, tol) integer, intent(out) :: type, txt, per, val real, intent(out) :: tol

Arguments

type
Pie chart annotation type

= GRADIAL	Radial
= GINTERNAL	Internal
= GEXTERNAL	External

txt
Flag determining whether text string is included in annotation

= GNOTEXT	Text string not included
= GTEXT	Text string included

per
Flag determining whether calculated percentage is included in annotation

= GNOPERCENT	Percentage value not included
= GPERCENT	Percentage value included

val
Flag determining whether data value is included in annotation

= GNODATA	Data value not included
= GDATA	Data value included

tol
Tolerance level. The minimum percentage of the whole Pie Chart that the segment must occupy before being annotated

Description The routine ggEnqPieChartAnnotation() returns the current Pie Chart annotation settings as set by the routine ggSetPieChartAnnotation(). If ggSetPieChartAnnotation() has not been called the default settings are returned.

See Also Page 151
ggSetPieChartAnnotation

ggEnqPieChartSettings

Syntax

```
C/C++:     void ggEnqPieChartSettings(float *radius, GPOINT *origin, float *angle);
```

```
F90:       subroutine ggEnqPieChartSettings(radius, origin, angle)

           real, intent(out) :: radius, angle
           type(GPOINT), intent(out) :: origin
```

Arguments

radius
Value giving the current radius of the Pie Chart in the current units (default units are millimetres)

origin.x
Value giving the current X coordinate of the centre of the Pie Chart in user space coordinates

origin.y
Value giving the current Y coordinate of the centre of the Pie Chart in user space coordinates

angle
Value giving the current start angle of the Pie Chart in degrees

Description The routine ggEnqPieChartSettings() returns the current Pie Chart settings as defined by the most recent calls to ggSetPieChartFrame() and ggSetPieChartStartAngle() (or the default settings).

See Also Page 151
ggSetPieChartStartAngle
ggSetPieChartFrame

ggEnqPlotFrame

Syntax

```
C/C++:    void ggEnqPlotFrame(int *flg, GLIMIT *limits);
```

```
F90:      subroutine ggEnqPlotFrame(flg, limits)

          integer, intent(out) :: flg
          type(GLIMIT), intent(out) :: limits
```

Arguments

flg
Flag to indicate source of drawing limits

= GGINOMODE	limits set by current GINO window
= GGINOGRAF	limits set by ggSetPlotFrame()

limits.xmin
The minimum limit of the graph drawing area in the horizontal direction

limits.xmax
The maximum limit of the graph drawing area in the horizontal direction

limits.ymin
The minimum limit of the graph drawing area in the vertical direction

limits.ymax
The maximum limit of the graph drawing area in the vertical direction

Description

The routine ggEnqPlotFrame() returns the current limits of the graph drawing area and a flag to indicate how they were set.

By default, the drawing area is defined as the current window limits as set by GINO. However, the routine ggSetPlotFrame() can be used to set alternative limits, without affecting the clipping limit.

See Also

Page 16
ggSetPlotFrame

ggEnqTextChartAttribs

Syntax

```
C/C++:    void ggEnqTextChartAttribs(float *width, float *height, int *jslcr, int *head, int *line);
```

```
F90:      subroutine ggEnqTextChartAttribs(width, height, jslcr, head, line)

          real, intent(out) :: width, height
          integer, intent(out) :: jslcr, head, line
```

Arguments

width
Column width in user space coordinates

height
Column height in user space coordinates

jslcr
Justification flag for column entry

= GLEFT	Left justification
= GCENTRE	Centre justification
= GRIGHT	Right justification

head
Header switch for Text Chart columns

= GNOHEAD	No header cell for column
= GHEAD	Header cell added at top of column

line
Text chart frame line style index

< 0	Switch frame box off
= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

Description

The routine ggEnqTextChartAttribs() returns the current settings of the Text Chart characteristics as set by the routine ggSetTextChartAttribs().

See Also

Page 154
ggSetTextChartAttribs

ggEnqValueAttribs

Syntax

C/C++: **void ggEnqValueAttribs**(int *xpos, int *ypos, float *xory, GPOINT *offset, float *angstr, int *jstmb, int *jslcr);

F90: **subroutine ggEnqValueAttribs**(xpos, ypos, xory, offset, angstr, jstmb, jslcr)

```
integer, intent(out) :: xpos, ypos, jstmb, jslcr
real, intent(out) :: xory, angstr
type(GPOINT), intent(out) :: offset
```

Arguments

xpos
Position of data value control point in horizontal direction

= GOUTSIDELEFT	left of lower value, left of data value
= GINSIDELEFT	right of lower value, left of data value
= GCENTRE	centre of area at data value
= GINSIDERIGHT	left of upper value, right of data value
= GOUTSIDERIGHT	right of upper value, right of data value
= GSPECIFIED	positioned at **xory**

ypos
Position of data value control point in vertical direction

= GOUTSIDEBOTTOM	below lower value below data value
= GINSIDEBOTTOM	above lower value, below data value
= GMIDDLE	middle of area at data value
= GINSIDETOP	below upper value above data value
= GOUTSIDETOP	above upper value, above data value
= GSPECIFIED	positioned at **xory**

xory
Position of data value control point in graphical coordinates, if **xpos** or **ypos** = GSPECIFIED

offset.x
X Offset from control point in user space coordinates

offset.y
Y Offset from control point in user space coordinates

angstr
Data value string angle

jstmb
Vertical justification of data value

= GTOP	Top justified - string below control point
= GMIDDLE	Middle justified - string centred at control point
= GBOTTOM	Bottom justified - string above control point

jslcr
Horizontal justification of data value

= GLEFT	Left justified
= GCENTRE	Centre justified
= GRIGHT	Right justified

Description

The routine ggEnqValueAttribs() returns the current settings of the value display attributes as set by the routine ggSetValueAttribs().

See Also

Page 66
ggSetValueAttribs

ggEnqVectorAttribs

Syntax

C/C++: **void ggEnqVectorAttribs**(int *pos, float *vecmin, float *vecmax, float *factor);

```
F90:    subroutine ggEnqVectorAttribs(pos, vecmin, vecmax, factor)

        integer, intent(out) :: pos
        real, intent(out) :: vecmin, vecmax, factor
```

Arguments

pos
Vector position flag

= GTAIL	Arrow tail
= GMIDDLE	Middle of arrow
= GHEAD	Head of arrow

vecmin
Absolute vector strength represented by a zero length vector

vecmax
Absolute vector strength represented by a unit length vector

factor
Overall vector length scaling factor

Description

The routine ggEnqVectorAttribs() returns the position and scaling attributes for Vector Charts as set by ggSetVectorAttribs().

See Also

Page 123
ggSetVectorAttribs

ggEnqVectorChartFrame

Syntax

```
C/C++:    void ggEnqVectorChartFrame(GLIMIT *limits);
```

```
F90:      subroutine ggEnqVectorChartFrame(limits)
          type(GLIMIT), intent(out) :: limits
```

Arguments

limits.xmin
Horizontal minimum of Vector Chart in Graphical coordinates

limits.xmax
Horizontal maximum of Vector Chart in Graphical coordinates

limits.ymin
Vertical minimum of Vector Chart in Graphical coordinates

limits.ymax
Vertical maximum of Vector Chart in Graphical coordinates

Description

The routine ggEnqVectorChartFrame() enquires the area in graphical coordinates onto which Vector Charts are mapped as set by ggSetVectorChartFrame(). The graphical coordinate system is set up by the latest calls to the routines ggSetAxesPos() and ggSetAxesScaling().

If ggSetVectorChartFrame() has not been called or if the routine ggRestoreVectorSettings() has been called the default area is returned. That is the area represented by the intersection of the limits of the horizontal (X) and vertical (Y) axes.

See Also

Page 123
ggSetAxesPos
ggSetAxesScaling
ggSetVectorChartFrame
ggAddVectors
ggRestoreVectorSettings

ggEnqVectorLimits

Syntax

```
C/C++:    void ggEnqVectorLimits(float *smin, float *smax);
```

```
F90:      subroutine ggEnqVectorLimits(smin, smax)
          real, intent(out) :: smin, smax
```

Arguments

smin
Minimum absolute strength of vector that may be displayed using ggAddVectors()

smax
Maximum absolute strength of vector that may be displayed using ggAddVectors()

Description

The routine ggEnqVectorLimits() returns the minimum and maximum absolute vector strength that may be drawn by subsequent calls to ggAddVectors() as set by ggSetVectorLimits().

If **smin** and **smax** are equal then clipping is switched off and all vectors with strength not equal to zero are drawn. The routine ggRestoreVectorSettings() also switches clipping off.

See Also

Page 123
ggSetVectorLimits
ggAddVectors
ggRestoreVectorSettings

ggFillAreaChart

Syntax

```
C/C++:    void ggFillAreaChart(int nareas, GAREACHART *areas, int xory, int *fill,
          int *line);
```

```
F90:      subroutine ggFillAreaChart(nareas, areas, xory, fill, line)

          integer, intent(in) :: nareas, xory, fill(*), line(*)
          type(GAREACHART), intent(in) :: areas(*)
```

Arguments

nareas
The number of areas to be plotted

areas
Array of structures giving data for the Area Chart

xory
Flag determining which axis the data ranges are shown on, and on which axis the heights are shown

= GXAXIS	Data on X axis, heights on Y axis
= GYAXIS	Data on Y axis, heights on X axis

fill
Integer array, of dimension **nareas**, determining the fill styles to be used to fill each bar

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
> 256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line
Integer array, of dimension **nareas**, determining the line style to be used to fill each column

= GCURRENT	Specifies the current line style
= 1 -256	Specifies the line style index
>256	Specifies the current line style

The value of an element of **line** is irrelevant where the corresponding element of **fill** has a value of less than -1

Description

The routine ggFillAreaChart() draws filled rectangles defined by the data ranges held in the components **areas.s** and **areas.f** and height values held in the components **areas.h1** and **areas.h2**, with respect to either the current axes, as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default axes used by one of the high level routines. The widths of the areas, held in **areas.s** and **areas.f**, are shown on the X axis if **xory**=GXAXIS, and on the Y axis if **xory**=GYAXIS.

The areas are filled in the style determined by the combination of the corresponding elements of **fill** and **line**. Where a **fill** element is equal to GHOLLOW, only the boundary of the area is drawn. One or more of the segments may be left unfilled by giving the corresponding element(s) of **fill** a value of -2 or less. Negative values of **line** cause a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

See Also

Page 108
ggPlotAreaChart
ggAddAreaChartOutline
ggAddAreaChartValues

ggFillBarChart

Syntax

```
C/C++:    void ggFillBarChart(int nbars, GBARCHART *bars, float frac, int *fill,
           int *line);
```

```
F90:      subroutine ggFillBarChart(nbars, bars, frac, fill, line)

          type(GBARCHART), intent(in) :: bars(*)
          integer, intent(in) :: nbars, fill(*), line(*)
          real, intent(in) :: frac
```

Arguments

nbars
The number of bars in the Bar Chart

bars
Array of dimension **nbars**, giving the start and finish values of all the bars in the Bar Chart

frac
Fraction of an interval to be filled for each bar

fill
Integer array, of dimension **nbars**, determining the fill styles to be used to fill each column

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}

= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
>256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line

Integer array, of dimension **nbars**, determining the line style to be used to fill each column

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

The value of an element of **line** is irrelevant where the corresponding element of **fill** has a value less than -1

Description

The routine ggFillBarChart() fills the bars of a Bar Chart, with respect to either the current axes, as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default axes used by ggPlotBarChart(). If a discrete axis has not been defined (using ggSetAxesScaling() with **scale**=GDISCRETE) or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

The area filled for each bar has the start and finish values defined in the components **bars.s** and **bars.f** and the width ((length of discrete axis)/**nbars**) * **frac**. If **frac** = 1.0, the fill occupies the whole interval.

The bars are centred on the tick marks on the discrete axis.

The bars are filled in the style determined by the combination of the corresponding elements of **fill** and **line**. Where a **fill** element is equal to GHOLLOW, only the boundary of the bar is drawn. One or more of the segments may be left unfilled by giving the corresponding element(s) of **fill** a value of -2 or less. Negative values of **line** cause a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

Where the boundaries of the bars are not drawn; if required, these can be produced by a call to ggAddBarChartOutline() or the complete Bar Chart routine ggPlotBarChart(). If ggPlotBarChart() and ggFillBarChart() are used together, ggPlotBarChart() should be called first to set up the axis system.

See Also

Page 96
ggPlotBarChart
ggAddBarChartOutline
ggAddBarChartValues

ggFillBelowDataset

Syntax

C/C++:	**void ggFillBelowDataset**(int npts, GPOINT *points, float xylev, int xory, int fill, int line);

F90:	**subroutine ggFillBelowDataset**(npts, points, xylev, xory, fill, line) type(GPOINT), intent(in) :: points(*) integer, intent(in) :: npts, xory, fill, line real, intent(in) :: xylev

Arguments

npts
The number of points on graph line

points
Array, of dimension **npts**, giving the data values of the points to be defined on the graph line

xylev
Value determining point on X or Y axis which is filled to. If **xylev** lies beyond the axes limits then that limit is used instead of **xylev**

xory
Flag determining whether **xylev** is on the X or Y axis

= GXAXIS	**xylev** on X axis
= GYAXIS	**xylev** on Y axis

fill
Fill style

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}

>256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line
Number determining the line style to be used to fill graph

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

The value of **line** is irrelevant where **fill** is less than -1

Description

The routine ggFillBelowDataset() fills the area between **npts** points on a graph held in the **points** array, and a line perpendicular to the X or Y axis, held in **xylev**. If **xylev** is outside of the current axes set up by the last axis definition then the relevant axis limits are used. Complete filled graphs may be achieved in conjunction with ggFillBetweenDatasets().

The area is filled in the style determined by the combination of the values of **fill** and **line**. Where **fill** is equal to GHOLLOW, only the boundary of the area is drawn. A negative value of **line** causes a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

See Also

Page 61
ggFillBetweenDatasets

ggFillBetweenDatasets

Syntax

```
C/C++:    void ggFillBetweenDatasets(int n1, GPOINT *xy1, int n2, GPOINT *xy2, int fill, int
          line);
```

```
F90:      subroutine ggFillBetweenDatasets(n1, xy1, n2, xy2, fill, line)

          integer, intent(in) :: n1, n2, fill, line
          type(GPOINT), intent(in) :: xy1(*), xy2(*)
```

Arguments

n1
The number of points on first graph line

xy1
Array, of dimension **n1**, giving the data values of the points to be defined on the first graph line

n2
The number of points on second graph line

xy2
Array, of dimension **n2**, giving the data values of the points to be defined on the second graph line

fill
Fill style

< -1	Specifies no fill and no boundary

= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
>256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line

Number determining the line style to be used to fill graph

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

The value of **line** is irrelevant where **fill** is less than -1

Description

The routine ggFillBetweenDatasets() fills the area between two sets of points on a graph held in the arrays **xy1** and **xy2**, where each set of points is made up of **n1** and **n2** points respectively. Complete filled graphs may be achieved in conjunction with ggFillBelowDataset().

The area is filled in the style determined by the combination of the values of **fill** and **line**. Where **fill** is equal to GHOLLOW, only the boundary of the area is drawn. A negative value of **line** causes a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

See Also

Page 61
ggFillBelowDataset

ggFillHistogram

Syntax

```
C/C++:    void ggFillHistogram(int ncols, float *yarray, float frac, int *fill, int *line);
```

```
F90:      subroutine ggFillHistogram(ncols, yarray, frac, fill, line)

          integer, intent(in) :: ncols, fill(*), line(*)
          real, intent(in) :: yarray(*), frac
```

Arguments

yarray
Array, of dimension **ncols**, giving the heights of all the columns in the Histogram

ncols
The number of columns in the Histogram

frac
Fraction of an interval to be filled for each column

fill
Integer array, of dimension **ncols**, determining the fill styles to be used to fill each column

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
>256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line
Integer array, of dimension **ncols**, determining the line style to be used to fill each column

= GCURRENT	Specifies the current line style

= 1 to 256	Specifies the line style index
>256	Specifies the current line style

The value of an element of **line** is irrelevant where the corresponding element of **fill** has a value less than -1

Description

The routine ggFillHistogram() fills the columns of a Histogram with respect to either the current axes, as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default Histogram axes used by ggPlotHistogram(). If a discrete axis has not been defined using ggSetAxesScaling(), or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis. The area filled for each column has the height defined in the array **yarray** and the width = ((length of X axis)/**ncols**) * **frac**.

If **frac** = 1.0, the fill occupies the whole interval. The columns are centred on the tick marks on the discrete axis.

The columns are filled in the style determined by the combination of the corresponding elements of **fill** and **line**. Where a **fill** element is equal to GHOLLOW, only the boundary of the column is drawn. One or more of the segments may be left unfilled by giving the corresponding element(s) of **fill** a value of -2 or less. Negative values of **line** cause a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

The column boundaries are not drawn; if required these can be produced by a call to ggAddHistogramOutline() or the high level routine ggPlotHistogram(). If ggPlotHistogram() and ggFillHistogram() are used together, ggPlotHistogram() should be called first to set up the axis system.

See Also

Page 90
ggPlotHistogram
ggAddHistogramOutline
ggAddHistogramValues

ggFillMultiHistogram

Syntax

C/C++:

```
void ggFillMultiHistogram(int type, float *rdata, int ndim1, int ncols, int ndata, float
frac, float gap, int line[], int is1, int is2);
```

F90:

```
subroutine ggFillMultiHistogram(type, rdata, ndim1, ncols, ndata, frac, gap, fill, line,
is1, is2)

integer, intent(in) :: type, ndim1, ncols, ndata
real, intent(in) :: rdata(ndim1,*), frac, gap
integer, intent(in) :: fill(*),line(*), is1, is2
```

Arguments

type
Type of multi-histogram chart

= GSTACKED	Data sets stacked in single column
= GCLUSTERED	Data sets displayed as multiple columns

rdata
Two dimensional array, of dimension **ndim1**,* giving the heights of the columns in each of the data sets

ndim1
The first dimension of the data array **rdata**

ncols
The number of compound columns, or clusters of columns to be drawn. This can be from 1 to **ndim1**

ndata
The number of data sets for each column or column cluster to be drawn. This can be from 1 to the second dimension of the data array **rdata**

frac
Fraction of an interval to be filled for each column or column cluster

gap
Size of gap between members of the cluster, as a fraction of the width of a single member of the cluster. Range 0.0 to 1.0 (only used for type GCLUSTERED)

fill
Integer array, of dimension **ndata**, determining the fill style to be used to fill each data set. The corresponding component of each column will be drawn with the same fill style

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
>256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line
Integer array, of dimension **ndata**, determining the line style to be used to fill each data set. The corresponding component of each column will be drawn with the same line style

= GCURRENT	Specifies the current line style

= 1 to 256	Specifies the line style index
>256	Specifies the current line style

is1
The start position in the first dimension of the data array to be used as the first item (category) on the discrete axis

is2
The start position in the second dimension of the data array to be used as the first component on the continuous axis (or first column of cluster)

Description

The routine ggFillMultiHistogram() fills the columns of a multi-data set Histogram with respect to the current axes as set up by ggSetAxesPos() and ggSetAxesScaling(). If a discrete axis has not been defined using ggSetAxesScaling(), or both axes have been defined as discrete axes, the discrete axis is assumed to be the X axis. Linear scaling is assumed as the default for the Y axis.

The routine is designed to display a stacked or clustered histogram representing a block of data from an arbitrarily sized two dimensional array - **rdata**. Where **ndim1** is the first dimension of the array and **is1** and **is2** specify the starting offset of the data block. The dimensions of the data being represented is **ncols** by **ndata**, where **ncols** is the number of data items in each set and **ndata** is the number of data sets.

The width of each stacked column or clustered column = ((length of the discrete axis)/**ncols**) * **frac** where a value of 0.99 will cause either the stacked column or clustered column to nearly touch the adjacent column. Values of **frac** outside the range 0.0 to 1.0 are clipped to 0.1 and 0.9 respectively with values between 0.5 and 0.9 giving the most satisfactory results.

All the columns of each data set [i] are filled in the fill style specified by **fill[i]** and the line style specified by **line[i]**.

See Also

Page 118
ggBlockFillHistogram
ggBlockFillMultiHistogram

ggFillStepChart

Syntax

C/C++:
```
void ggFillStepChart(int nsteps, GSTEPCHART *steps, float base, int xory, int *fill, int
*line);
```

F90:
```
subroutine ggFillStepChart(nsteps, steps, base, xory, fill, line)
integer, intent(in) :: nsteps, xory, fill(*), line(*)
type(GSTEPCHART), intent(in) :: steps(*)
real, intent(in) :: base
```

Arguments

nsteps
The number of steps to be plotted

steps
Array of dimension **nsteps**, giving the start and finish widths and height values of all the steps in the Step Chart

base
Base value in graphical coordinates which steps are filled to

xory
Flag determining which axis the data ranges are shown on, and on which axis the heights are shown

= GXAXIS	Data on X axis, heights on Y axis
= GYAXIS	Data on Y axis, heights on X axis

fill
Integer array, of dimension **nsteps**, determining the fill styles to be used to fill each column

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
>256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line
Integer array, of dimension **nsteps**, determining the line style to be used to fill each column

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

The value of an element of **line** is irrelevant where the corresponding element of **fill** has a value less than -1

Description

The routine ggFillStepChart() draws filled rectangles defined by the data ranges held in the components **steps.s** and **steps.f** and between the **base** value on the axis on which the **steps.h** values are measured and the height values held in the component **steps.h**. The display of the rectangles is drawn with respect to either the current axes, as set up by ggSetAxesPos() and ggSetAxesScaling(), or the default axes used by one of the high level routines. The widths of the columns, held in **steps.s** and **steps.f**, are shown on the X axis if **xory**=GXAXIS, and on the Y axis if **xory**=GYAXIS.

The rectangles are filled in the style determined by the combination of the corresponding elements of **fill** and **line**. Where a **fill** element is equal to GHOLLOW, only the boundary of the rectangle is drawn. One or more of the rectangles may be left unfilled by giving the corresponding element(s) of **fill** a value of -2 or less. Negative values of **line** cause a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

See Also

Page 102

ggMoveToGraphPoint

Syntax

```
C/C++:    void ggMoveToGraphPoint(float x, float y);
```

```
F90:      subroutine ggMoveToGraphPoint(x, y)
          real, intent(in) :: x, y
```

Arguments

x
Value giving the X part of the graphical axes coordinate to which the pen will be moved

y
Value giving the Y part of the graphical axes coordinate to which the pen will be moved

Description

The routine ggMoveToGraphPoint() moves the pen to the point (X, Y), either inside or outside the graph limits, in the graphical axes system set up by the last axis definition calls or by one of the axis control routines.

The point need not be within the area defined by the axes. The pen is moved to the position (X, Y) without drawing a line.

See Also

Page 167, 168
ggAddGraphLine
ggDrawArrow

ggPlotAreaChart

Syntax

```
C/C++:    void ggPlotAreaChart(int nareas, GAREACHART *areas, int scx, int scy);
```

```
F90:      subroutine ggPlotAreaChart(nareas, areas, scx, scy)

          integer, intent(in) :: nareas, scx, scy
          type(GAREACHART), intent(in) :: areas(*)
```

Arguments

nareas
The number of areas to be plotted

areas
Array of dimension **nareas**, containing data for all the areas in the Area Chart

scx
The type of scaling used on the X axis

= GLINEAR	Linear scale on axis
= GLOG10	Log_{10} scale on axis

scy
The type of scaling used on the Y axis

= GLINEAR	Linear scale on axis
= GLOG10	Log_{10} scale on axis

Description

The routine ggPlotAreaChart() draws a frame to fit the available drawing area or current window and plots an Area Chart within it. The areas are drawn as rectangles defined by the data ranges held in the components **areas.s** and **areas.f** and height values held in the components **areas.h1** and **areas.h2**. The axes are scaled according to the values of **scx** and **scy**.

The default axes annotation may be changed using the routines ggSetAxesAnnotation() and/or ggSetAxesAttribs().The default data ranges and axes intervals can be changed using the routine ggSetGraphScaling().

See Also

Page 82
ggFillAreaChart
ggAddAreaChartOutline
ggAddAreaChartValues
ggSetAxesAnnotation
ggSetAxesAttribs
ggSetGraphScaling
ggSetPlotFrame

ggPlotBarChart

Syntax

C/C++:
```
void ggPlotBarChart(int nbars, GBARCHART *bars, float frac, int scx, int scy, float
vbeg, float vend);
```

F90:
```
subroutine ggPlotBarChart(nbars, bars, frac, scx, scy, vbeg, vend)

type(GBARCHART), intent(in) :: bars(*)
integer, intent(in) :: nbars, scx, scy
real, intent(in) :: frac, vbeg, vend
```

Arguments

nbars
The number of bars to be plotted

bars
Array of dimension **nbars**, giving the start and finish values of the bars to be plotted

scx
Set the X axis as the discrete (perpendicular to bars) axis, or the type of scaling on the continuous axis (axis defining bar length)

= GLINEAR	Linear scaling on the X axis

= GLOG10	Logarithmic scaling on the X axis
= GDISCRETE	The X axis is the discrete axis

scy
Set the Y axis as the discrete (perpendicular to bars) axis, or the type of scaling on the continuous axis (axis defining bar length)

= GLINEAR	Linear scaling on the Y axis
= GLOG10	Logarithmic scaling on the Y axis
= GDISCRETE	The Y axis is the discrete axis

frac
Fraction of an interval to be occupied by each bar

vbeg
The centre value of the first bar on the discrete (perpendicular to bars) axis

vend
The centre value of the last bar on the discrete (perpendicular to bars) axis

Description

The routine ggPlotBarChart() draws a frame to fit the available drawing area or current window and plots a Bar Chart within it. The bars on the Bar Chart have the width:

((length of discrete axis) / **nbars**) * **frac**.

and have the start and end positions defined in the **bars** array. If **frac** = 1.0, only the necessary lines are drawn, ie, lines common to two bars are omitted. If **frac** = 0.0, two coincident lines are drawn centred on the tick mark. If **frac** < 0.0, the default value is 0.0 and if **frac** > 1.0, the default value is 1.0.

The Bar Chart is automatically scaled and annotated to make maximum use of the drawing area. If **vbeg** = **vend**, no numeric annotation is written on the discrete axis. The type of scaling of the continuous axis and the orientation of the discrete axis are defined by the value given in **scx** & **scy**.

The default axes annotation may be changed using the routines ggSetAxesAnnotation() and/or ggSetAxesAttribs().

See Also

Page 76
ggSetAxesAnnotation
ggSetAxesAttribs
ggFillBarChart
ggAddBarChartOutline
ggAddBarChartValues
ggSetPlotFrame

ggPlotGraph

Syntax

C/C++:	**void ggPlotGraph**(int npts, GPOINT *points, int scx, int scy, int style, int axis);

```
F90:    subroutine ggPlotGraph(npts, points, scx, scy, style, axis)

        integer, intent(in) :: npts, scx, scy, style, axis
        type(GPOINT), intent(in) :: points(*)
```

Arguments

npts
The number of points to be plotted

points
Array of dimension **npts**, giving the X and axis values of the points to be defined on the graph

scx
The type of scaling used on the X axis

= GLINEAR	Linear scale on X axis
= GLOG10	Log_{10} scale on X axis

scy
The type of scaling used on the Y axis

= GLINEAR	Linear scales on Y axis
= GLOG10	Log_{10} scale on Y axis

style
Type of graph

= GSYMBOLS	Symbol drawn at each data point
= GSTRAIGHT	Points drawn with straight lines
= GCUBIC	Points joined with piecewise smooth parametric cubic curve
= GAKIMA	Point joined with smooth Akima curve
= GSPLINE	Points joined with parametric cubic spline

Negative values of the above will add a symbol at each point.

axis
Type of axes

= GFRAME	A frame is drawn
= GAXES	Axes are drawn through origin of data (if present) or at the bottom left corner of drawing area

Description

The routine ggPlotGraph() is a general purpose graph drawing routine that plots the data supplied in the components **points.x** and **points.y** according to the above arguments. The graph is drawn to fit the available drawing area or current window. Scaling and annotation are performed automatically.

The resultant curve drawn with **style**=GCUBIC or GSPLINE can extend outside the graph frame or drawing area. The curve end conditions may be set by using the routines ggSetCurveStartConds() and/or ggSetCurveEndConds().

Graphs with **style**= GSYMBOLS or GSTRAIGHT, can be affected by the current missing value mode as set by the routine ggDefineMissingValues().

The default axes annotation may be changed using the routines ggSetAxesAnnotation() and/or ggSetAxesAttribs().The default data ranges and axes intervals can be changed using the routine ggSetGraphScaling().

See Also

Page 48
ggDefineMissingValues
ggSetAxesAnnotation
ggSetAxesAttribs
ggSetCurveStartConds
ggSetCurveEndConds
ggSetGraphScaling
ggSetPlotFrame

ggPlotHistogram

Syntax

C/C++:	**void ggPlotHistogram**(int ncols, float *yarray, float frac, int scy, float vbeg, float vend);

F90:	**subroutine ggPlotHistogram**(ncols, yarray, frac, scy, vbeg, vend) integer, intent(in) :: ncols, scy real, intent(in) :: yarray(*), frac, vbeg, vend

Arguments

ncols
The number of columns to be plotted

yarray
Array, of dimension **ncols**, giving the heights of the columns to be plotted on the Histogram

frac
Fraction of an interval to be occupied by each column

scy
The type of scaling used on the Y axis

= GLINEAR	Linear scale on Y axis
= GLOG10	Log_{10} scale on Y axis

vbeg
Value specifying the centre value of the first column on the X axis

vend
Value specifying the centre value of the last column on the X axis

Description

The routine ggPlotHistogram() draws a frame to fit the available drawing area or current window and plots a Histogram within it. The columns on the Histogram have the width:

((length of X axis)/**ncols**) * **frac**.

If **frac** = 1.0, only the necessary verticals are drawn, ie, lines common to two columns are omitted. If **frac** = 0.0, two coincident vertical lines are drawn centred on the tick mark. If **frac** ≤ 0.0, the default value is 0.0 and if **frac** ≥ 1.0, the default value is 1.0. Each column has the height defined in array **yarray**.

The Histogram is automatically scaled and annotated to make maximum use of the drawing area. If **vbeg** = **vend**, no numeric annotation is written on the X axis. The type of scaling on the continuous axis is defined by the value given in **scy**. The default axes annotation may be changed using the routines ggSetAxesAnnotation() and/or ggSetAxesAttribs().

See Also

Page 72
ggSetAxesAnnotation
ggSetAxesAttribs
ggSetPlotFrame
ggFillHistogram
ggAddHistogramValues

ggPlotPieChart

Syntax

```
C/C++:    void ggPlotPieChart(int nsegs, float *value, char *string[ ], int *fill, int *line);
```

```
F90:      subroutine ggPlotPieChart(nsegs, value, string, fill, line)

          integer, intent(in) :: nsegs, fill(*), line(*)
          real, intent(in) :: value(*)
          character*(*), intent(in) :: string(*)
```

Arguments

nsegs
The number of segments to be plotted (between 1 and 50)

value
Array, of dimension **nsegs**, of segment values for the chart. These need not be percentages as ggPlotPieChart() calculates the percentage value of each segment automatically

string
Array of dimension **nsegs**, holding the labels for the segments

fill
Integer array, of dimension **nsegs**, determining the fill styles to be used to fill each segment

< -1	Specifies no fill and no boundary
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}

= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
>256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line

Integer array, of dimension **nsegs**, determining the line style to be used to fill each segment

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

The value of an element of **line** is irrelevant where the corresponding element of **fill** has a value less than -1

Description

The routine ggPlotPieChart() draws a complete annotated, filled Pie Chart. The Pie Chart is drawn to fit the available drawing window, or with respect to the Pie Chart frame defined by the most recent call to ggSetPieChartFrame(). The segments are drawn anticlockwise from the three o'clock position, or from the start angle defined by the most recent call to ggSetPieChartStartAngle(). The absolute value for each segment takes a proportion of the pie. The proportion is defined as the segment value divided by the sum of all the segment values.

Each segment of the Pie Chart consists of the following elements; the background filling, the annotation and associated box and the segment boundary.

The segments are filled in the style determined by the combination of the corresponding elements of **fill** and **line**. Where a **fill** element is equal to GHOLLOW, only the boundary of the segment is drawn. One or more of the segments may be left unfilled by giving the corresponding element(s) of **fill** a value of -2 or less. Negative values of **line** cause a warning to be output and the absolute value is used. The default line styles, hatch styles and fill styles appear in Appendix A of this manual. The current line style is left unchanged.

The default annotation for the Pie Chart is to print the segment label horizontally in a masked box within the segment boundary. Other forms of annotation are available including radial and external, each of which may include a combination of the segment label, the percentage value of the segment and the data value itself. All these options are set with the Pie Chart annotation routine ggSetPieChartAnnotation(). The routine ggSetPieChartBoxType() controls the filling/masking and drawing of the annotation box for internal segment annotation.

By default the segment boundaries are drawn in the current pen colour, these can be switched off using ggSetPieChartBoundSwitch().

Individual segments may be extracted from the Pie Chart centre using the routine ggSetPieChartExplosion().

See Also

Page 140
ggSetPlotFrame
ggSetPieChartStartAngle
ggSetPieChartAnnotation
ggSetPieChartBoxType
ggSetPieChartBoundSwitch
ggSetPieChartExplosion
ggSetPieChartFrame

ggPlotStepChart

Syntax

C/C++:	**void ggPlotStepChart**(int nsteps, GSTEPCHART *steps, float base, int scx, int scy, int drop);

F90:	**subroutine ggPlotStepChart**(nsteps, steps, base, scx, scy, drop) integer, intent(in) :: nsteps, scx, scy, drop type(GSTEPCHART), intent(in) :: steps(*) real, intent(in) :: base

Arguments

nsteps
The number of steps to be plotted

steps
Array of dimension **nsteps**, giving the start and finish widths and height values of all the areas in the Step Chart

scx
The type of scaling used on the X axes

= GLINEAR	Linear scale on axis
= GLOG10	Log_{10} scale on axis

scy
The type of scaling used on the Y axes

= GLINEAR	Linear scales on axis
= GLOG10	Log_{10} scale on axis

base
Base value in graphical coordinates which steps are drawn to depending on the value of **drop**

drop
Flag determining how step edges are drawn

= GDROPTYPE0	Step heights only are drawn
= GDROPTYPE1	Link adjacent steps

= GDROPTYPE2	Link adjacent steps and draw non-adjacent edges to **base**
= GDROPTYPE3	Draw all step edges to **base**

Description

The routine ggPlotStepChart() draws a frame to fit the available drawing area or current window and plots a Step Chart within it. The steps are drawn as columns with widths determined by the values in **steps.s** and **steps.f**, and with heights of the values held in the component **steps.h**.

The columns may be represented in one of four ways depending on the value of **drop**. If **drop** = GDROPTYPE2 or GDROPTYPE3 then all or the end column edges are drawn to **base** on the Y axis.

The axes are scaled according to the value of **scx, scy**.

The default axes annotation may be changed using the routines ggSetAxesAnnotation() and/or ggSetAxesAttribs().The default data ranges and axes intervals can be changed using the routine ggSetGraphScaling().

See Also

Page 79
ggAddStepChartOutline
ggAddStepChartValues
ggFillStepChart
ggSetAxesAnnotation
ggSetAxesAttribs
ggSetGraphScaling
ggSetPlotFrame

ggPlotXYPolarChart

Syntax

```
C/C++:    void ggPlotXYPolarChart(int npts, GPOINT *points, int line);
```

```
F90:      subroutine ggPlotXYPolarChart(npts, points, line)

          integer, intent(in) :: npts, line
          type(GPOINT), intent(in) :: points
```

Arguments

npts
The number of points to be plotted

points
Array of dimension **npts**, giving the X and Y values of the points to be defined on the graph

line
The type of line to be drawn

= GSYMBOLS	Symbol drawn at each data point
= GSTRAIGHT	Points drawn with straight lines (default)
= GCUBIC	Points joined with piecewise smooth parametric cubic curves

= GAKIMA	Point joined with smooth Akima curve
= GSPLINE	Points joined with parametric cubic splines

Negative values of the above will add a symbol at each point

Description

The routine ggPlotXYPolarChart() draws a complete Polar Chart to fit the available drawing area or current window and plots a graph within it from the data held in the components **points.x** and **points.y**. The type of line joining the data points is determined by **line**. The polar plot is automatically scaled and annotated to fit centrally within the available drawing area, giving the full 360 degrees radius and containing all the points in the data array.

Graphs with **style**= GSYMBOLS or ±GSTRAIGHT, can be affected by the current missing value mode as set by the routine ggDefineMissingValues().

The default axes annotation may be changed using the routines ggSetAxesAnnotation() and/or ggSetAxesAttribs().

See Also

Page 127
ggDefineMissingValues
ggSetAxesAnnotation
ggSetAxesAttribs
ggSetPlotFrame

ggRestoreAxesSettings

Syntax

C/C++:	**void ggRestoreAxesSettings**(void);
F90:	**subroutine ggRestoreAxesSettings**

Arguments None

Description

The routine ggRestoreAxesSettings() restores the default settings for axis positioning, scaling, and annotation. ggRestoreAxesSettings() restores all the attributes set using the routines ggSetAxesPos(), ggSetAxesScaling(), ggSetDateAxesAnnotation(), ggSetDateAxesScaling(), ggSetPolarChartAttribs(), ggSetAxesAnnotation(), and ggSetAxesAttribs().

ggRestoreAxesSettings() also switches off any graphical drawing boundary set up with the routine ggSetPlotFrame(), such that future axis positioning for any of the complete graph or chart routines is calculated to fit the current GINO drawing area or window.

See Also

Page 15, 23, 45 and 135
ggSetAxesPos
ggSetAxesScaling
ggSetAxesAnnotation
ggSetAxesAttribs
ggSetDateAxesAnnotation
ggSetPlotFrame
ggSetPolarChartAttribs
ggSetDateFormat
ggSetDateAxesScaling

ggRestoreBlockChartAttribs

Syntax

C/C++:	**void ggRestoreBlockChartAttribs**(void);

F90:	**subroutine ggRestoreBlockChartAttribs**

Arguments

None

Description

The routine ggRestoreBlockChartAttribs() restores the Block Chart attributes to their default settings.

A full description of the attributes and their default values is documented with the setting routine ggSetBlockChartAttribs().

See Also

Page 88
ggEnqBlockChartAttribs
ggSetBlockChartAttribs

ggRestorePieChartSettings

Syntax

C/C++:	**void ggRestorePieChartSettings**(void)**;**

F90:	**subroutine ggRestorePieChartSettings**

Arguments

None

Description

The routine ggRestorePieChartSettings() restores the default Pie Chart frame, in which the size and position of the Pie Chart are calculated to fit the available drawing area or current window. This resets any frame set up by ggSetPieChartFrame() or the GINOGRAF drawing area set by ggSetPlotFrame().

ggRestorePieChartSettings() also restores all Pie Chart attributes to their defaults. These include:

Start angle (ggSetPieChartStartAngle())

Annotation settings (ggSetPieChartAnnotation())

Annotation box type (ggSetPieChartBoxType())

Segment boundary switch (ggSetPieChartBoundSwitch())

Explosion factors (ggSetPieChartExplosion())

See Also

Page 151
ggSetPlotFrame
ggSetPieChartStartAngle
ggSetPieChartAnnotation
ggSetPieChartBoxType
ggSetPieChartBoundSwitch
ggSetPieChartExplosion
ggSetPieChartFrame

ggRestoreVectorSettings

Syntax

```
C/C++:    void ggRestoreVectorSettings(void);
```

```
F90:      subroutine ggRestoreVectorSettings
```

Arguments None

Description

The routine ggRestoreVectorSettings() resets the clipping, positioning and scaling attributes of Vector Chart arrows to their default state. The different attributes are set by the routines ggSetVectorChartFrame(), ggSetVectorLimits() and ggSetVectorAttribs().

ggRestoreVectorSettings() restores their state to that at GINOGRAF initialization.

See Also

Page 123
ggSetVectorChartFrame
ggSetVectorLimits
ggSetVectorAttribs

ggReturnLineCoeffs

Syntax

```
C/C++:    void ggReturnLineCoeffs(int type, int npts, GPOINT *points, int ncoef, float coeffs[ ],
          int *nmax, int *er);
```

```
F90:      subroutine ggReturnLineCoeffs(type, npts, points, ncoef, coeffs, nmax, er)

          integer, intent(in) :: type, npts, ncoef
          integer, intent(out) :: nmax, er
          type(GPOINT), intent(in) :: points(*)
          real, intent(out) :: coeffs(*)
```

Arguments

type
Type of line fitting performed on the data set

= GLEASTSQUARE	Least squares straight line fit
> 1	Reserved for future use

points
Array of dimension **npts**, giving the X and Y axis values of the points defined on the graph

npts
The number of points in X and Y

ncoef
The number of coefficients required to be returned

coeffs
Array of dimension **ncoef**, returning the coefficients of the fitted line

nmax
The total number of coefficients of the fitted line

= 0,	No coefficients returned due to argument or fitting error
> 0,	Number of coefficients of successful fit

er
Error flag

= GSUCCESS	Successful fit
= GFAIL	Unsuccessful fit

Description

The routine ggReturnLineCoeffs() takes a set of data points and fits a straight line or curve to it, returning their coefficients to the user. No graphics output is generated by this routine.

The data is supplied in the components **points.x**, **points.y** each having **npts** points. The coefficients are returned in the array **coeffs** which should have the same dimension as the number of coefficients required as set in **ncoef**. The actual number of coefficients calculated by the fitting algorithm is returned in **nmax** which may be more or less than **ncoef**.

In the case of the straight line fit (**type**=GLEASTSQUARE), the two coefficients ‘a’ and ‘b’ of line ‘y=ax + b’ are returned in **coeffs**(2) and **coeffs**(1) respectively.

If less than two points are supplied or less that one coefficient is requested, an error message is generated and no fitting is attempted. If there is enough data but a fit is not possible (ie, if the points are located in one position, or if all the points have the same Y coordinate values) the error flag **er** is set to 1 but no error message is produced. In either of these cases **nmax** is set to zero to indicate that no coefficients have been returned.

See Also

Page 169

ggSetAxesAnnotation

Syntax

```
C/C++:    void ggSetAxesAnnotation(int ndp, int npower, int asty, int xory);
```

```
F90:      subroutine ggSetAxesAnnotation(ndp, npower, asty, xory)
          integer, intent(in) :: ndp, npower, asty, xory
```

Arguments

ndp
The number of decimal places

< 0, **ndp** decimal places are always displayed

> 0, Up to **ndp** decimal places are displayed
= 0, No decimal places are used forcing rounding to the nearest integer value

ndp must be in the range -9 to 9

npower
The power to which annotation is raised, e.g. x10$^{\mathbf{npower}}$. **npower** must be in the range -15 to 15, any value outside this range will result in default annotation

asty
Axis scale format

= GNOSCALE	No scale factor is displayed
= GSCALEPOWEROF10	Scale factor is displayed as *10^n
= GSCALEZEROS	Scale factor is displayed as ‘000 or 0.00’
= GSCALEWORD	Scale factor is displayed in words
= GSCALEPREFIX	Scale factor is displayed in engineering units

Scale types GSCALEWORD and GSCALEPREFIX require that **npower** is a multiple of 3

xory
Specified axis

= GXAXIS	Parameters set for X axis
= GYAXIS	Parameters set for Y axis

Description

The routine ggSetAxesAnnotation() is used to alter the format of all numeric annotation and the style of the scale factor on linear graphical axes.

Its main use is with axes annotation format for all the complete graph and chart routines as well as the component axis drawing routines ggDrawAxes(), ggDrawPolarAxes() and ggAddGrid(). The default format is described under ggDrawAxes().

ndp can be used to optionally control or force the display of the number of decimal places if there are more or less than two decimal places. When **ndp** is positive, the axis annotation will only contain up to that number of decimal places as required by the calculated increment.

NB. Certain machines may not have the capability to cope with numbers to an accuracy of nine decimal places. This may produce undesirable results.

npower is used to control the scale factor of the annotation forcing it to a particular value in the range -15 to 15. If **npower** is outside this range, GINOGRAF will display the actual power factor of the data supplied.

asty provides the five alternatives in the display of the scale factor at the end of the axis. If set to GNOSCALE, NO scale factor is displayed at the end of the axes. Obviously, this should be used with care as the values displayed at the tick marks could be misleading and require a scale factor. Type GSCALEPOWEROF10 is the default, displaying the power factor. Type GSCALEZEROS will display the scale factor with the required number of 0’s, for example:

x10^4 is displayed as ‘0000
x10^{-3} is displayed as 0.00’

Types GSCALEWORD and GSCALEPREFIX are used when **npower** is a multiple of 3 displaying the following words:

	GSCALEWORD	**GSCALEPREFIX**
x10^{-15}	Quadrillionths	atto
x10^{-12}	Trillionths	pico
x10^{-9}	Billionths	nano
x10^{-6}	Millionths	micro
x10^{-3}	Thousandths	milli
x10^{3}	Thousand	kilo
x10^{6}	Million	mega
x10^{9}	Billion	giga
x10^{12}	Trillion	tera
x10^{15}	Quadrillion	peta

If **npower** is not a multiple of 3, for type GSCALEWORD or GSCALEPREFIX, the default scale type will be used and a warning message output. If the power factor is outside the range -15 to 15 or if ggSetAxesAnnotation() has not been called, GINOGRAF will select an appropriate scale factor which is zero or a multiple of 3.

ggSetAxesAnnotation() is also used to control numerical annotation on Value Charts, Pie Charts and Text Charts. Value charts use the format of the relevant axis, but Pie Charts and Text Charts use the attributes set for the Y axis (**xory** = GYAXIS) so that values can be matched to those displayed on the Y axis if necessary. The user should note however, that these routines do not provide a means for displaying the scale factor and so in most cases the value of **npower** should be zero unless a direct relationship can be made with a Y axis and its scale factor.

All the annotation attributes are set to their respective defaults if the routine ggRestoreAxesSettings() is called.

See Also

Page 27, 145 and 157
ggDrawAxes
ggRestoreAxesSettings
ggEnqAxesAnnotation
ggDrawPolarAxes
ggAddGrid

ggSetAxesAttribs

Syntax

C/C++:	**void ggSetAxesAttribs**(int swi, float xy, int nstart, int nskip, float aoff, float angstr, int jstmb, int jslcr, int reduc, int xory);

F90:	**subroutine ggSetAxesAttribs**(swi, xy, nstart, nskip, aoff, angstr, jstmb, jslcr, reduc, xory) integer, intent(in) :: swi, nstart, nskip, jstmb, jslcr, reduc, xory real, intent(in) :: xy, aoff, angstr

Arguments

swi

Annotation position

= GONAXIS	On specified axis
= GOFFSET	Positioned at **xory**

xy

The position in user space coordinates of the annotation in either the X or Y direction. **xy** is only used if **swi**=GOFFSET

nstart

Tick mark number at which the annotation starts

nskip

Number of annotation elements to be skipped during annotation

< 0	Automatic skip generation
= GNONE	No labels skipped - all annotation elements are output
> 0	Skip **nskip** elements

aoff

Offset as a proportion of distance between major tick marks on the specified axis

angstr

Annotation string angle

jstmb

Vertical justification for each annotation element

= GBOTTOM	Bottom justified - string above control point
= GMIDDLE	Middle justified - string centred at control point
= GTOP	Top justified - string below control point
= GDEFAULTPOSITION	Default for requested axis

jslcr

Horizontal justification for each annotation element

= GLEFT	Left justified
= GCENTRE	Centre justified
= GRIGHT	Right justified
= GDEFAULTPOSITION	Default for requested axis

reduc

Annotation character size reduction switch

= GNOREDUCE	No size reduction
= GREDUCE	Reduce annotation so that no elements overlap (only operates when **nskip** = GNONE)

xory

Specified axis

= GXAXIS	Parameters set for X axis
= GYAXIS	Parameters set for Y axis

Description

The routine ggSetAxesAttribs() sets general characteristics for text and numerical axis annotation produced by all the complete graph and chart routines as well as the component axis drawing routines ggDrawAxes(), ggDrawAxesLabels(), ggDrawPolarAxes() and ggAddGrid().

The arguments **nstart** and **nskip** determine the first major tick mark to be annotated and whether any elements are to be skipped. If **nskip** < 0, elements are automatically skipped if there is not enough space between tick marks to output the requested elements.

The annotation is output with respect to a control point which is positioned in line with each major tick mark to the specified axis. Where the annotation is output ‘on’ the axis (**swi**=0), the control point is twice the tick mark length away from the axis on either the clockwise or anti-clockwise side. (The side being determined by the relevant parameter in the output routine ggAddGrid(), ggDrawAxes() or ggDrawAxesLabels()). Where the annotation is output at **xy** the control point is on the specified line.

The control point can then be shifted using the offset factor **aoff** which is measured as a proportion of the distance between the major tick marks on the relevant axis. The annotation elements may be rotated by **angstr** about the control point and justified in both vertical and horizontal directions using **jstmb** and **jslcr**. The justification is performed with respect to the angle that the annotation is output.

When all annotation elements are being output (**nskip**=GNONE) and their bounding boxes overlap then the character size may be reduced by setting **reduc** to GREDUCE. The character size of the annotation along the whole axis is reduced in height and width in order to retain the current aspect ratio.

All the axes attributes for both axes are set to their respective defaults if the routine ggRestoreAxesSettings() is called.

See Also

Page 31
ggDrawAxes
ggRestoreAxesSettings
ggDrawAxesLabels
ggEnqAxesAttribs
ggDrawPolarAxes
ggAddGrid

ggSetAxesPos

Syntax

```
C/C++:    void ggSetAxesPos(int or, float xor, float yor, float axlen, int xory);
```

```
F90:      subroutine ggSetAxesPos(or, xor, yor, axlen, xory)

          integer, intent(in) :: or, xory
          real, intent(in) :: xor, yor, axlen
```

Arguments

or

Flag determining the position of the point (**xor**, **yor**) on the axis

= GDATAORIGIN	(**xor**, **yor**) represents the point at which the natural origin should occur on the drawing area

= GAXISSTART	(**xor**, **yor**) represents the point where the axis starts on the drawing area

xor
Value which specifies the X part of the user space coordinates defining the position of the axis in the drawing area

yor
Value which specifies the Y part of the user space coordinates defining the position of the axis in the drawing area

axlen
Value specifying the length of the axis in the current units (default units are millimetres). Units can be changed using the GINO routine gDefinePictureUnits(), described in the GINO User Guide

xory
Specified axis

= GXAXIS	The X axis is defined
= GYAXIS	The Y axis is defined

Description

The routine ggSetAxesPos() defines the position and length of an axis in user space coordinates. **or** determines whether the coordinates (**xor**, **yor**) represent the start of the axis or the natural origin. If the natural origin is not within the range of the data, the axis starts at the coordinate (**xor**, **yor**). If graphical drawing begins before ggSetAxesPos() is called, suitable defaults are calculated in terms of the available drawing area or current window.

Successive calls to ggSetAxesPos() for the same axis override earlier definitions. Coordinates defined in terms of axes set up by ggSetAxesPos() and ggSetAxesScaling() are referred to as graphical axes coordinates.

The effects of ggSetAxesPos() can be cancelled using ggRestoreAxesSettings(), allowing defaults overridden by ggSetAxesPos() to be restored.

See Also

Page 20
ggRestoreAxesSettings
ggEnqAxesPos

ggSetAxesScaling

Syntax

C/C++: **void ggSetAxesScaling**(int scale, int nints, float vbeg, float vend,int xory);

F90: **subroutine ggSetAxesScaling**(scale, nints, vbeg, vend, xory)

```
integer, intent(in) :: scale, nints, xory
real, intent(in) :: vbeg, vend
```

Arguments

scale
The type of scaling to be used

= GLINEARTYPE1	Linear scaling automatically calculated from the available axis length. Results in a range which includes **vbeg** and **vend** as closely as possible, divided into approximately **nints** intervals
= GLINEARTYPE2	Linear scaling automatically calculated to produce precisely **nints** intervals in a range including **vbeg** and **vend**
= GLINEARTYPE3	Linear scaling selected by the user with precisely **nints** intervals, **vbeg** on the first interval and **vend** on the last interval
= GLOG10	Log_{10} scaling
= GDISCRETE	Discrete axis with **nints** columns or rows and tick marks at the column or row centres

nints
The number of intervals on a linear axis, or the number of columns or rows on a discrete axis. The value of **nints** is irrelevant when used with logarithmic scaling

vbeg
Value specifying the beginning of the range to be included on the axis

vend
Value specifying the end of the range to be included on the axis

xory
Specified axis

= GXAXIS	The X axis is defined
= GYAXIS	The Y axis is defined

Description

The routine ggSetAxesScaling() defines the scaling characteristics of the last **xory** axis defined by ggSetAxesPos(), or the default axis if ggSetAxesPos() has not been called. If graphical drawing begins before ggSetAxesScaling() is called, the following defaults are used:

scale = GLINEARTYPE3
nints = 9
vbeg = 1.0
vend = 10.0

If, for **scale** = GLINEARTYPE1, GLINEARTYPE2, GLINEARTYPE3, or GDISCRETE, **vbeg** = **vend**, the following defaults are used:

new **vbeg** = **vbeg** - 0.5 (exponent of **vbeg**)
new **vend** = **vend** + 0.5 (exponent of **vend**)

If, for **scale** = GLOG10, **vbeg** = **vend**, the following default is used:

vbeg = **vbeg**/10
vend = 10 * **vend**

The actual number of intervals and ranges calculated by ggSetAxesScaling() may be enquired through the routine ggEnqAxesScaling().

Successive calls to ggSetAxesScaling() for a particular axis override previous calls for that axis. Coordinates defined in terms of axes set up by ggSetAxesScaling() and ggSetAxesPos() are referred to as graphical axes coordinates.

The effects of ggSetAxesScaling() may be overridden using ggRestoreAxesSettings(), allowing parameters set by ggSetAxesScaling() to be restored to their default values.

See Also

Page 21
ggSetAxesPos
ggRestoreAxesSettings
ggEnqAxesScaling

ggSetBlockChartAttribs

Syntax

C/C++:	**void ggSetBlockChartAttribs**(int coloff, float azim, float elev, float depth, float top, float side);

F90:	**subroutine ggSetBlockChartAttribs**(coloff, azim, elev, depth, top, side) integer, intent(in) :: coloff real, intent(in) :: azim, elev, depth, top, side

Arguments

coloff
Colour index offset for shading (deafult = 20)

azim
Azimuth angle of block in range -60.0 to 60.0 (default = 30.0)

elev
Elevation angle of block in range -60.0 to 60.0 (default = 30.0)

depth
Depth as fraction of column width in range 0.1 to 10.0 (default = 1.0)

top
The relative lightness of the top of a column compared to that of the front in range 0.0 to 100.0 (default = 0.67)

side
The relative lightness of the side of a column compared to that of the front in range 0.0 to 100.0 (default = 0.33)

Description

The routine ggSetBlockChartAttribs() sets the Block Chart attributes for all the Block Chart routines.

The colour index offset is used for the allocation of colour table entries to set the top and side shading colours. The first index used is **coloff**+1. Each of the standard block chart routines require two indices where as the multi-histogram routine requires 2 * the number of groups indices. User should be aware that the colour table entries **coloff**+1 and beyond are modified by the block filling routines and are not reset by any GINOGRAF routine.

The azimuth and elevation angles determine the shape of the block filling. When both are positive, the filling is to the right and above the front of the column or area. Negative values switch the block shading to the alternative side.

The depth fraction (**depth**) is used as specified above in the histogram and bar chart routines, but because the width is variable in the step and area charts, **depth** is multiplied by a fixed value (10% of the specified axis length) to determine the nominal block depth.

Both arguments **top** and **side** may be set to make the top and side panels of the block either darker or lighter than the front. i.e Values less than 1.0 will produce darker colours and values greater than 1.0 will produce lighter colours. In all cases the hue of the front panel is maintained.

The default block chart attributes may be restored using the routine ggRestoreBlockChartAttribs().

See Also

Page 87
ggBlockFillAreaChart
ggBlockFillBarChart
ggBlockFillHistogram
ggBlockFillMultiHistogram
ggBlockFillStepChart
ggEnqBlockChartAttribs
ggRestoreBlockChartAttribs

ggSetCurveEndConds

Syntax

```
C/C++:    void ggSetCurveEndConds(int fin, float cosfin, float sinfin, float xfin, float yfin);
```

```
F90:      subroutine ggSetCurveEndConds(fin, cosfin, sinfin, xfin, yfin)

          integer, intent(in) :: fin
          real, intent(in) :: cosfin, sinfin, xfin, yfin
```

Arguments

fin

End conditions for end of subsequent graph curve drawing routine

= GXPOINT	Direction of curve defined such that it would pass through the extra point **xfin**, **yfin**
= GNONE	No end condition set
= GANGLE	Direction angle of curve defined in terms of **cosfin** and **sinfin**

cosfin

Cosine of finish angle

sinfin

Sine of finish angle

xfin

X coordinate of extra point in graph coordinates

yfin
Y coordinate of extra point in graph coordinates

Description

The routine ggSetCurveEndConds() sets the finish conditions of the next curve drawing routine used by GINOGRAF (ggPlotGraph(), ggAddGraphCurve(), ggAddAkimaCurve() or ggAddGraphSpline()). The end conditions may be defined in terms of an extra point (in graph coordinates) or an angle.

The specified end conditions remain in effect for all curve drawing routines, until reset with **fin** being set to GNONE.

See Also

Page 55
ggSetCurveStartConds
ggAddAkimaCurve
ggAddGraphCurve
ggPlotGraph
ggAddGraphSpline

ggSetCurveStartConds

Syntax

C/C++:	**void ggSetCurveStartConds**(int beg, float cosbeg, float sinbeg, float xbeg, float ybeg);

```
F90:      subroutine ggSetCurveStartConds(beg, cosbeg, sinbeg, xbeg, ybeg)

          integer, intent(in) :: beg
          real, intent(in) :: cosbeg, sinbeg, xbeg, ybeg
```

Arguments

beg
End conditions for start of subsequent graph curve drawing routine

= GXPOINT	Direction of curve defined such that it would pass through the extra point **xbeg**, **ybeg**
= GNONE	No start condition set
= GANGLE	Direction angle of curve defined in terms of **cosbeg** and **sinbeg**

cosbeg
Cosine of start angle

sinbeg
Sine of start angle

xbeg
X coordinate of extra point in graph coordinates

ybeg
Y coordinate of extra point in graph coordinates

Description

The routine ggSetCurveStartConds() sets the start conditions of the next curve drawing routine used by GINOGRAF (ggPlotGraph(), ggAddGraphCurve(), ggAddAkimaCurve() or ggAddGraphSpline()). The start conditions may be defined in terms of an extra point (in graph coordinates) or an angle.

The specified start conditions remain in effect for all curve drawing routines, until reset with **beg** being set to GNONE.

See Also

Page 55
ggSetCurveEndConds
ggAddAkimaCurve
ggAddGraphCurve
ggPlotGraph
ggAddGraphSpline

ggSetDateAxesAnnotation

Syntax

C/C++:	**void ggSetDateAxesAnnotation**(int fdow, int fday, int fmon,int fyear, int xory);

F90:	**subroutine ggSetDateAxesAnnotation**(fdow, fday, fmon, fyear, xory) integer, intent(in) :: fdow,fday,fmon,fyear,xory

Arguments

fdow
Format for Day of Week

< 0,	Alphanumeric with -**fdow** characters (up to 9)
= GNONE	Not present
= 1,	Numeric (1 to 7) with 1 = Monday

fday
Format for Day

< 0,	Alphanumeric (1st, 2nd, 3rd, etc.)
= GNONE	Not present
= 1,	Numeric (1 to 31)

fmon
Format for Month

< 0,	Alphanumeric with -**fmon** characters (up to 9)
= GNONE	Not present
= 1,	Numeric (1 to 12)

fyear
Format for Year

= GNONE	Not present
= 2,	Two digit display
= 4,	Four digit display

xory
Specified axis

= GXAXIS	The X axis is defined
= GYAXIS	The Y axis is defined

Description Defines the output format for date axis annotation for use in conjunction with the date axis format and scaling routines ggSetDateAxesAnnotation() and ggSetDateAxesScaling().

Each of the day of the week, day and month may be specified as alphanumeric, not present or numeric in the label attached to the major tick mark of the date axis. The year element may be omitted or contain 2 or 4 digits. Where an alphanumeric format is specified, for days of the week and months, the component is truncated to the specified number of characters. ie. where **fmon** is set to -3, months are output as Jan, Feb, Mar, etc.

The ordering of the date components is set with the output format settings of the routine ggSetDateFormat().

The routine ggRestoreAxesSettings() will reset the date format to the default settings for both axes.

See Also Page 40
ggRestoreAxesSettings
ggSetDateFormat
ggSetDateAxesScaling

ggSetDateAxesScaling

Syntax

```
C/C++:    void ggSetDateAxesScaling(int scale, int dincr, char dbeg[ ],
          char dend[ ], int xory);
```

```
F90:      subroutine ggSetDateAxesScaling(scale, dincr, dbeg, dend, xory)

          integer, intent(in) :: scale, dincr, xory
          character*(*), intent(in) :: dbeg, dend
```

Arguments

scale

The type of scaling to be used

= GLINEARTYPE1	Results in a range which includes **dbeg** and **dend** rounded up to the nearest suitable date increment (**dincr** is not used)
= GLINEARTYPE2	As type 1 but uses the supplied date increment
= GLINERATYPE3	Using the supplied **dincr**, the range will start at dbeg and end on a date increment at or beyond **dend**

dincr

The increment used for major tick marks on the date axis

= GDECADE	Decade (10 years)
= GYEAR	Year
= GMONTH	Month
= GWEEK	Week (7 days starting on Monday)
= GDAY	Day

dbeg

Date for the beginning of the range to be included on the axis

dend
Date for end of the range to be included on the axis

xory
Specified axis

= GXAXIS	The X axis is defined
= GYAXIS	The Y axis is defined

Description

The routine ggSetDateAxesScaling() defines a date scaling type for the **xory** axis replacing any numerical scaling type defined by ggSetAxesScaling(). The position set by ggSetAxesPos() is unaltered.

Rather than specifying the number of increments, ggSetDateAxesScaling() specifies the increment type to be used on the major tick marks of the date axis. Minor tick marks will represent the next smaller increment (if there is space to display them)

Note that axes tickmarks are always equally spaced and therefore the positioning of month and year tickmarks and their associated label may not be drawn in exactly the correct position due to the varying length of these increments.

The start and end dates are supplied as character strings of up to 10 characters, formatted according to the current date input format type as set up by ggSetDateFormat(). Dates which are incorrectly formatted are interpreted as January 1st 1900.

Data represented as a series of dates in the same format may be converted to the appropriate real day values required for the data display routines using the routine ggConvertDates().

Numerical scaling can be reset on an axis using ggSetAxesScaling() or ggRestoreAxesSettings(), the latter of which will reset numerical scaling to their default values.

See Also

Page 39
ggSetAxesPos
ggSetAxesScaling
ggRestoreAxesSettings
ggConvertDates
ggSetDateFormat

ggSetDateFormat

Syntax

```
C/C++:    void ggSetDateFormat(int inform, char insep, int ouform, char ousep);
```

```
F90:      subroutine ggSetDateFormat(inform, insep, ouform, ousep)

          integer, intent(in) :: inform, ouform
          character, intent(in) :: insep, ousep
```

Arguments

inform
Date format for input dates

= GBRITISH	British form dd/mm/yy or dd/mm/yyyy

= GAMERICAN	American form mm/dd/yy or mm/dd/yyyy
= GLOGICAL	Standard Logical form yy/mm/dd or yyyy/mm/dd

insep
Date separator for input dates

ouform
Date format for output dates

= GBRITISH	British form dd/mm/yy or dd/mm/yyyy
= GAMERICAN	American form mm/dd/yy or mm/dd/yyyy
= GLOGICAL	Standard Logical form yy/mm/dd or yyyy/mm/dd

ousep
Date separator for output dates

Description

Sets the date format for numeric date character strings. Essentially this routine controls the ordering of the date components and their separator when in numeric format.

The input format setting is used by ggConvertDates(), ggConvertDateToGraph() and ggSetDateAxesScaling() routines. In these routine, dates are supplied as a character string of up to 10 characters with the defined input separator separating the day, month and year. Days and months can be given as one or two digits, there can be a leading zero if the value is less than 10. Years can be supplied as two or four digits, with two digits being taken to apply to the years 1950 to 2049, ie. 05 means 2005 not 1905. The input format is also used in the routines ggEnqDateAxesScaling() and ggConvertGraphToDate().

The output format is used to control the ordering of the date components on date axes although their full format is controlled by the routine ggSetDateAxesAnnotation(). The output separator is only used for fully numeric dates.

If either format type is out of range, then the default British format is set.

See Also

Page 39
ggSetDateAxesAnnotation
ggConvertDates
ggSetDateAxesScaling

ggSetGraphCharMode

Syntax

C/C++:	**void ggSetGraphCharMode**(int sw);

F90:	**subroutine ggSetGraphCharMode**(sw) integer, intent(in) :: sw

Arguments

sw
Flag determining whether software transformable character strings are to be used within GINOGRAF

= GGINOMODE	Use current GINO character mode
= GSOFTWARE	Set software transformable character strings

Description

The routine ggSetGraphCharMode() sets a switch to determine whether software transformable characters or the current GINO character mode are to be used for all character output by GINOGRAF.

By default GINOGRAF sets software transformable characters so that graph annotation is correctly positioned if any GINO transformations are current. If **sw** = GGINOMODE, GINOGRAF uses the current character mode for all its character output which allows the use of hardware characters if desired.

sw should not be set to GGINOMODE if GINO transformations are current because annotation will be incorrectly positioned.

The current GINO character mode is always restored at the end of each GINOGRAF routine.

See Also

Page 14

ggSetGraphScaling

Syntax

```
C/C++:      void ggSetGraphScaling(int mode);
```

```
F90:        subroutine ggSetGraphScaling(mode)
            integer, intent(in) :: mode
```

Arguments

mode

Graph scaling mode

= GDEFAULT	Default
= GEQUALLIMITS	Equal limits for both axes
= GEQUALRANGES	Equal ranges for both axes
= GEQUALGRAPHINTERVALS	Equal sized interval in graph coordinates
= GEQUALSPACEINTERVALS	Equal sized intervals in space coordinates (Square grid)

Description

The routine ggSetGraphScaling() sets the scaling mode for the complete graph, step chart and area chart drawing routines.

By default, all the complete graph and chart drawing routines, calculate the separate ranges of X and Y from the data supplied to the routine and display axes representing these ranges. The routine ggSetGraphScaling() can be used to modify the final output of the graph or chart by altering the ranges or intervals on the axes according to the required mode.

Note that, in GEQUALSPACEINTERVALS mode, the length of one of the axes may be reduced in order to comply with the request for equal sized intervals in space coordinates.

See Also

Page 86
ggPlotAreaChart
ggPlotGraph
ggPlotStepChart

ggSetGridMarker

Syntax

C/C++:	**void ggSetGridMarker**(int sym);

F90:	**subroutine ggSetGridMarker**(sym) integer, intent(in) :: sym

Arguments

sym
GINO symbol number

= GSPOT	}
= GUP	}
= GDOWN	}
= GPLUS	}
= GCROSS	} Standard GINO symbols
= GBOX	}
= GDIAMOND	}
= GCIRCLE	}
= GSTAR	}
= 9 - 23,	Optional hardware symbol
> 23	Character from font table

Description

The routine ggSetGridMarker() defines the grid intersection symbol. The default being a cross symbol displayed at the intersection of the major tick marks.

sym may be any GINO symbols as described in the GINO user guide.

See Also

Page 26
ggAddGrid

ggSetPieChartAnnotation

Syntax

C/C++:	**void ggSetPieChartAnnotation**(int type, int txt, int per, int val, float tol);

F90:	**subroutine ggSetPieChartAnnotation**(type, txt, per, val, tol) integer, intent(in) :: type, txt, per, val real, intent(in) :: tol

Arguments

type
Pie chart annotation type

= GRADIAL	Radial
= GINTERNAL	Internal
= GEXTERNAL	External

txt
Flag determining whether text string is included in annotation

= GNOTEXT	Text string not included
= GTEXT	Text string included

per
Flag determining whether calculated percentage is included in annotation

= GNOPERCENT	Percentage value not included
= GPERCENT	Percentage value included

val
Flag determining whether data value is included in annotation

= GNODATA	Data value not included
= GDATA	Data value included

tol
Tolerance level. The minimum percentage of the whole Pie Chart that the segment must occupy before being annotated

Description

The routine ggSetPieChartAnnotation() sets the Pie Chart annotation type for the complete Pie Chart routines ggPlotPieChart() and ggAddPieChartSegment().

The style and position of the annotation is set by **type**. Radial annotation (GRADIAL) outputs a single string, bisecting the Pie Chart segment in such a way that it is readable from left to right when viewed from the 6 o'clock position (Y axis negative direction). Internal annotation (GINTERNAL) is printed horizontally in a masked box within the segment. However, if the required strings do not fit then they are automatically printed outside the segment (as for external annotation). External annotation (GEXTERNAL) prints the required output as a single string horizontally outside the segment with a connecting line.

The annotation may consist of up to three elements: text strings held in the array **string**, percentage values calculated from the proportion of the whole pie that each segment occupies, and data values held in the array **value**. The inclusion of each of these elements is determined by the three flags **txt**, **per** and **val**.

Pie chart annotation can be suppressed for small segments by setting the value of **tol** to a suitable value. **tol** is measured as a percentage of a complete circle, therefore if it were set to 1.0 all segments less than 1% of the complete circle (3.6 degrees) would not be annotated.

Percentage values are always followed by a percent symbol. Value output is in the format set by ggSetAxesAnnotation() for the Y axis which by default is with up to 2 decimal places. The value may also include prefix and suffix strings set by ggSetValueTags().

The routine ggSetPieChartBoxType() is used to control the form of the box used with the internal Pie Chart annotation (GINTERNAL).

All the annotation settings are returned to their default with the routine ggRestorePieChartSettings().

See Also

Page 142
ggSetAxesAnnotation
ggSetPieChartBoxType
ggPlotPieChart
ggRestorePieChartSettings
ggAddPieChartSegment
ggSetValueTags

ggSetPieChartBoundSwitch

Syntax

C/C++:	**void ggSetPieChartBoundSwitch**(int switch);

F90:	**subroutine ggSetPieChartBoundSwitch**(int switch) integer, intent(in) :: switch

Arguments

switch

Pie chart boundary switch

= GOFF	Switch Pie Chart boundaries off
= GON	Pie chart boundaries are drawn

Description

The routine ggSetPieChartBoundSwitch() switches the drawing of Pie Chart boundaries on or off when using the Pie Chart routines ggPlotPieChart() and ggAddPieChartSegment(). The default condition is to draw the boundaries for all Pie Chart segments.

The boundary switch is restored to its default setting by the routine ggRestorePieChartSettings().

See Also

Page 148
ggPlotPieChart
ggRestorePieChartSettings
ggAddPieChartSegment

ggSetPieChartBoxType

Syntax

C/C++:	**void ggSetPieChartBoxType**(int type, int fill, int line);

F90:	**subroutine ggSetPieChartBoxType**(type, fill, line) integer, intent(in) :: type, fill, line

Arguments

type

Pie chart internal annotation box type

= GNONE	No filling/masking and no boxes

= GFILLED	Filling/masking done but no boxes drawn
= GBOXED	No filling/masking but box outlines are drawn

GFILLED and GBOXED can be combined by an OR operation to produce a box which is both filled and outlined.

fill

Fill style of Pie Chart annotation boxes for types GFILLED and GBOXED

< -1	Specifies box is masked only
= GHOLLOW	Specifies boundary only
= GSOLID	Specifies a solid fill
= GFINEHORIZONTAL	}
= GFINEVERTICAL	}
= GFINELEFTDIAGONAL	}
= GFINERIGHTDIAGONAL	}
= GFINEHORIZONTALGRID	}
= GFINEDIAGONALGRID	}
= GFINEHORIZONTALMESH	} Specifies the hatch
= GFINEDIAGONALMESH	} style index
= GCOARSEHORIZONTAL	}
= GCOARSEVERTICAL	}
= GCOARSELEFTDIAGONAL	}
= GCOARSERIGHTDIAGONAL	}
= GCOARSEHORIZONTALGRID	}
= GCOARSEDIAGONALGRID	}
= GCOARSEHORIZONTALMESH	}
= GCOARSEDIAGONALMESH	}
>256	Specifies a solid fill for software fill, or the fill style index for hardware fill

line

Line style to be used for filling boxes for types GFILLED and GBOXED

= GCURRENT	Specifies the current line style
= 1 to 256	Specifies the line style index
>256	Specifies the current line style

The value of **line** is irrelevant where **fill** has a value less than -1

Description

The routine ggSetPieChartBoxType() controls the form of the internal annotation boxes (where ggSetPieChartAnnotation() has been set to type GINTERNAL) for the complete Pie Chart routines ggPlotPieChart() and ggAddPieChartSegment(). ggSetPieChartBoxType() affects the annotation boxes of all the label types wherever they are placed within the segment area.

The value of **type** determines whether the annotation box is filled/masked, and whether a box outline drawn. If output, the box outline is drawn in the current GINO colour. If the box area is not filled/masked (**type** = GNONE) then the annotation is displayed over any segment filling. If **type** = GFILLED, the box area is filled in the specified fill and line style, except if **fill** < -1, where the box area is left unfilled.

If the value of **type** is out or range, a warning message is output and the default value is used.

The routine ggRestorePieChartSettings() resets the box type and fill styles back to their defaults.

See Also

Page 147
ggSetPieChartAnnotation
ggPlotPieChart
ggRestorePieChartSettings
ggAddPieChartSegment

ggSetPieChartExplosion

Syntax

```
C/C++:    void ggSetPieChartExplosion(int num, int *list, float *factor);
```

```
F90:      subroutine ggSetPieChartExplosion(num, list, factor)

          integer, intent(in) :: num, list(*)
          real, intent(in) :: factor(*)
```

Arguments

num

The number of segments to be drawn as exploded segments (not greater than 50)

= 0,	Lists of indices and explosion factors are ignored and all segments are drawn from the centre
> 0,	The **num** segments indexed in array **list** are drawn with the explosion factors in array **factor**

list

Array of dimension **num**, giving the indices of the segments to be drawn as exploded segments

factor

Array of dimension **num**, defining explosion factors

Description

The routine ggSetPieChartExplosion() defines a list of explosion factors, allowing the selective explosion of Pie Charts sectors. If ggSetPieChartExplosion() is called with **num** > 0, the **num** segments indexed in the array **list** will be exploded in all Pie Charts subsequently drawn by ggPlotPieChart(). For example, if **list** contains the values 1, 3 and 4, the first, third and fourth segments drawn will be extracted.

The distance from the centre of the Pie Chart to the inner point of the segment indexed in **list**(N) is then **factor**(N) * radius, all segments not indexed in **list** being drawn from the centre with no explosion factor.

Pie chart radius is automatically adjusted to take account of the explosion factor if the default Pie Chart frame is used. If ggSetPieChartFrame() has been called the user should ensure that exploded sections remain within the available drawing area or current window.

If ggSetPieChartExplosion() is called with **num** = 0, the lists of segment indices and explosion factors are deleted and all segments will be drawn from the centre in all Pie Charts subsequently drawn by ggPlotPieChart().

Once called, ggSetPieChartExplosion() remains active until called again with **num** = 0.

See Also

Page 148
ggPlotPieChart

ggSetPieChartFrame

Syntax

C/C++:	**void ggSetPieChartFrame**(float radius, float xcen, float ycen);

F90:	**subroutine ggSetPieChartFrame**(radius, xcen, ycen) real, intent(in) :: radius, xcen, ycen

Arguments

radius
Value giving the radius of the Pie Chart in the current units (default units are millimetres)

xcen
Value giving the X coordinate of the centre of the Pie Chart in user space coordinates

ycen
Value giving the Y coordinate of the centre of the Pie Chart in user space coordinates

Description

The routine ggSetPieChartFrame() defines a Pie Chart frame with radius **radius** and centre at the point (**xcen**,**ycen**). The default Pie Chart frame, used if ggSetPieChartFrame() is not called before drawing a Pie Chart, gives a radius and a centre calculated to fit the available drawing area or current window.

If ggSetPieChartFrame() has been used, automatic sizing and positioning can be restored using the routine ggRestorePieChartSettings(). The frame can be redefined at any point in the user program by another call to ggSetPieChartFrame().

See Also

Page 139
ggRestorePieChartSettings

ggSetPieChartStartAngle

Syntax

C/C++:	**void ggSetPieChartStartAngle**(float angle);

F90:	**subroutine ggSetPieChartStartAngle**(angle) real, intent(in) :: angle

Arguments

angle
Number specifying the start angle in degrees

Description

The routine ggSetPieChartStartAngle() defines the angle in degrees, measured anticlockwise from the three o’clock (positive X axis) position, to be used as the start point for the first defined segment when a Pie Chart is drawn using one of the routines ggPlotPieChart(). The default start angle used if ggSetPieChartStartAngle() is not called before drawing a Pie Chart is 0.0 degrees.

The routine ggRestorePieChartSettings() restores the start angle to 0.0 degrees.

See Also

Page 148
ggPlotPieChart
ggRestorePieChartSettings

ggSetPlotFrame

Syntax

```
C/C++:    void ggSetPlotFrame(GLIMIT *limits);
```

```
F90:      subroutine ggSetPlotFrame(limits)
          type(GLIMIT), intent(in) :: limits
```

Arguments

limits.xmin
The minimum limit of the graph drawing area in the horizontal direction

limits.xmax
The maximum limit of the graph drawing area in the horizontal direction

limits.ymin
The minimum limit of the graph drawing area in the vertical direction

limits.ymax
The maximum limit of the graph drawing area in the vertical direction

Description

The routine ggSetPlotFrame() defines a boundary for the GINOGRAF drawing area when using complete graph or chart routines.

By default, the drawing area is defined as the current window limits as set by GINO. ggSetPlotFrame() can be used therefore, to define an alternative drawing area, without affecting the clipping limit.

The drawing area affects the default position and length of graph axes as well as the placement of pie and polar charts. ggSetPlotFrame() does not restrict the manual placement of axes, pie and Polar Charts using the low level routines described in the relevant chapters.

The drawing area limits can be reset to match the current GINO window limits by calling ggRestoreAxesSettings() or ggRestorePieChartSettings().

If either of the minimum limits is greater than the corresponding maximum limit the two arguments are interchanged. If either of the horizontal or vertical extents of the drawing limits are zero a warning message is output and the default limits are used.

See Also

Page 16
ggRestoreAxesSettings
ggEnqPlotFrame
ggRestorePieChartSettings

ggSetPolarChartAttribs

Syntax

C/C++:	**void ggSetPolarChartAttribs**(float xorp, float yorp, float radlen, int scale);

```
F90:    subroutine ggSetPolarChartAttribs(xorp, yorp, radlen, scale)

        real, intent(in) :: xorp, yorp, radlen
        integer, intent(in) :: scale
```

Arguments

xorp

Value which specifies the X part of the user space coordinates defining the position of the origin of the axis in the drawing area

yorp

Value which specifies the Y part of the user space coordinates defining the position of the origin of the axis in the drawing area

radlen

Value specifying the radius of the polar plot in the current user space units

scale

The type of scaling to be used. In relation to the number of intervals (**nints**) and end of range value (**vendp**) as defined in ggDrawPolarAxes().

= GLINEARTYPE1	Linear scaling automatically calculated from the defined axis length (in ggDrawPolarAxes()). Results in a range from zero and which includes **vendp** as closely as possible, divided into approximately **nintp** intervals.
= GLINEARTYPE2	Linear scaling automatically calculated to produce precisely **nintp** intervals in a range from zero and including **vendp**.
= GLINEARTYPE3	Linear scaling selected by the user with precisely **nintp** intervals, zero on the first interval and **vendp** on the last interval.

Description

The routine ggSetPolarChartAttribs() defines the positioning, size and type of scaling for a polar plot. The coordinates (**xorp**, **yorp**) represent the origin of the polar plot in user space coordinates.

If ggSetPolarChartAttribs() is not called, the following defaults are used:

xorp,**yorp**	Centre of drawing area
radlen	0.375 * minimum of X or Y drawing area dimension
scale	3

See Also

Page 133
ggDrawPolarAxes

ggSetTextChartAttribs

Syntax

C/C++:	**void ggSetTextChartAttribs**(float width, float height, int jslcr, int head, int line);

```
F90:    subroutine ggSetTextChartAttribs(width, height, jslcr, head, line)

        real, intent(in) :: width, height
        integer, intent(in) :: jslcr, head, line
```

Arguments

width
Column width in user space coordinates

height
Column height in user space coordinates

jslcr
Justification flag for column entry

= GLEFT	Left justification
= GCENTRE	Centre justification
= GRIGHT	Right justification

head
Header switch for Text Chart columns

= GNOHEAD	No header cell for column
= GHEAD	Header cell added at top of column

line
Text chart frame line style index

< 0	Switch frame box off
= GCURRENT	Specifies the current line style
= 1 to 256,	Specifies the line style index
>256	Specifies the current line style

Description

The routine ggSetTextChartAttribs() defines the general characteristics for the text chart routines. The size of the Text Chart column is set by **width** and **height** which are in user space coordinates. ggSetTextChartAttribs() takes the absolute values supplied in the arguments. If either the **width** and/or **height** = 0.0 then a default size of column is defined, which is 10 times the character width and twice the character height times the number of rows required respectively. The character dimensions are those current at the time the Text Chart is output.

jslcr sets the justification of text or value entries of Text Chart columns which may be set to left, centre, or right justified. Left justified strings start half a character width away from the left hand side of the column and right justified strings end half a character width away from the right hand side of the column.

If **head** = GHEAD a header cell is added to the top of the column thus increasing the number of rows by one and the header string supplied in the Text Chart output routine is displayed centrally in that cell. If **head** = GNOHEAD, no header cell is added and the header string supplied in the Text Chart output routine is ignored.

Column frame boxes are drawn in the line style set by the value of **line**. If **line** < 0 then no column frame is drawn. Filled rectangles, line styles and symbols within Text Chart columns have their own line style settings passed as arguments. All other Text Chart elements, ie, header strings, text strings, values and percentages are displayed in the current GINO line style.

See Also

Page 154
ggEnqTextChartAttribs

ggSetValueAttribs

Syntax

C/C++:	**void ggSetValueAttribs**(int xpos, int ypos, float xory, float xoff, float yoff, float angstr, int jstmb, int jslcr);

F90:	**subroutine ggSetValueAttribs**(xpos, ypos, xory, xoff, yoff, angstr, jstmb, jslcr) integer, intent(in) :: xpos, ypos, jstmb, jslcr real, intent(in) :: xory, xoff, yoff, angstr

Arguments

xpos
Position of data value control point in horizontal direction

= GOUTSIDELEFT	left of lower value, left of data value
= GINSIDELEFT	right of lower value, left of data value
= GCENTRE	centre of area, at data value
= GINSIDERIGHT	left of upper value, right of data value
= GOUTSIDERIGHT	right of upper value, right of data value
= GSPECIFIED	positioned at **xory**

ypos
Position of data value control point in vertical direction

= GOUTSIDEBOTTOM	below lower value, below data value
= GINSIDEBOTTOM	above lower value, below data value
= GMIDDLE	middle of area, at data value
= GINSIDETOP	below upper value, above data value
= GOUTSIDETOP	above upper value, above data value
= GSPECIFIED	positioned at **xory**

xory
Position of data value control point in graphical coordinates, if **xpos** or **ypos** = GSPECIFIED

xoff, yoff
Offset from control point in user space coordinates

angstr
Data value string angle measured in degrees (anti-clockwise) from the 3 o'clock position

jstmb
Vertical justification of data value

= GBOTTOM	Bottom justified - string above control point
= GMIDDLE	Middle justified - string centred at control point
= GTOP	Top justified - string below control point

jslcr
Horizontal justification of data value

= GLEFT	Left justified
= GCENTRE	Centre justified
= GRIGHT	Right justified

Description

The routine ggSetValueAttribs() sets the attributes for the display of data values output by the ggAddxxxValues() routines. By default, values are positioned centrally over the coordinate position of the value itself (for ggAddGraphValues()) or in the centre of the column or area being annotated (for ggAddAreaChartValues(), ggAddBarChartValues(), ggAddHistogramValues(), ggAddStepChartValues()).

xpos and **ypos** (together with **xory**) set alternative positions of a control point about which the offset, angle and justification are taken. For **xpos** = GINSIDERIGHT / GINSIDELEFT / GOUTSIDERIGHT / GOUTSIDELEFT or **ypos** = GINSIDEBOTTOM / GINSIDETOP / GOUTSIDEBOTTOM / GOUTSIDETOP the control point is positioned one character width or height away from the upper or lower limit of the column or area. If **xpos** or **ypos** equals GSPECIFIED the control point is positioned at the specified **xory** coordinate on the respective axis. Where **xpos** or **ypos** is set to GSPECIFIED on a discrete axis (ie, where ggAddBarChartValues() or ggAddHistogramValues() are being used), the default (**xpos** or **ypos**=GCENTRE / GMIDDLE) is assumed. If both **xpos** and **ypos** are equal to GSPECIFIED for ggAddGraphValues(), ggAddAreaChartValues() or ggAddStepChartValues(), **all** the values are displayed at X=**xory**, Y=**xory**.

As there are no upper and lower limits when the routine ggAddGraphValues() is used, where **xpos** and **ypos** are negative, the control point is positioned to the left or below the coordinate position and where positive values of **xpos** and **ypos** are used (except GSPECIFIED) the control point is positioned to the right or above the coordinate position. The control point is effectively shifted by a character width or a character height in the respective direction away from each of the coordinate positions.

The control point can be shifted by an additional amount using the offsets **xoff** and **yoff**. These values are in user space coordinates and provide a means of finely adjusting the position of the data values.

The data value strings may be rotated by **angstr** about the control point and justified in either a vertical or horizontal direction using **jstmb** and **jslcr**. The justification is performed with respect to the angle that the data value is output.

Values may be appended by a prefix and/or suffix string to improve annotation with the routine ggSetValueTags(). If strings are appended, the whole string is treated as a single entity for justification purposes.

See Also

Page 64
ggEnqValueAttribs
ggSetValueTags

ggSetValueTags

Syntax

C/C++:	**void ggSetValueTags**(char prefix[ ], char suffix[ ]);

F90:	**subroutine ggSetValueTags**(prefix, suffix) character*(*), intent(in) :: prefix, suffix

Arguments

prefix
Prefix string

suffix
Suffix string

Description

Certain GINOGRAF routines, when outputting values, allow prefix and suffix strings to be output positioned in front of and after the values. The routine ggSetValueTags() allows the strings to be set up before use.

Where strings can be output with the value, the form of the value's output is set using ggSetAxesAnnotation(); outputting prefix and suffix strings does not change these settings.

Where no string is required for one of the parameters a blank string or the GINO string terminator should be used "*.".

See Also

Page 66, 92 and 158
ggSetAxesAnnotation

ggSetVectorAttribs

Syntax

C/C++:	**void ggSetVectorAttribs**(int pos, float vecmin, float vecmax, float factor);

F90:	**subroutine ggSetVectorAttribs**(pos, vecmin, vecmax, factor) integer, intent(in) :: pos real, intent(in) :: vecmin, vecmax, factor

Arguments

pos
Vector position flag

= GTAIL	Arrow tail
= GMIDDLE	Middle of arrow
= GHEAD	Head of arrow

vecmin
Absolute vector strength represented by a zero length vector

vecmax
Absolute vector strength represented by a unit length vector

factor
Overall vector length scaling factor

Description

The routine ggSetVectorAttribs() sets up the position and scaling attributes for Vector Charts drawn with the routine ggAddVectors().

Vector positions are located at the intersection of each grid point on the Vector Chart. The value of **pos** determines which part of the arrow is located at this grid position.

By default, each vector is scaled such that the maximum absolute vector strength is represented by an arrow of one unit grid length. That is the minimum distance between grid intersection points such that the arrows do not overlap. However, the arguments **vecmin** and **vecmax** can be used to define an alternative scaling where **vecmin** is the absolute strength of a vector which is represented by a zero length arrow (ie, not output) and **vecmax** is the absolute strength of a vector represented by an arrow of one unit grid length. If **vecmin** and **vecmax** are both set to the same value then the default scaling is restored.

vecmin and **vecmax** may be set inside or outside the actual range of vector strengths enabling complete flexibility of vector scaling.

Following the vector scaling calculated using **vecmin** and **vecmax**, an additional overall scale factor may be applied to each vector using the absolute value of **factor**.

If **vecmin** > **vecmax** the two values are inter-changed. If **factor** = 0.0 then no arrows are displayed.

See Also

Page 122
ggSetVectorChartFrame
ggSetVectorLimits
ggAddVectors
ggRestoreVectorSettings

ggSetVectorChartFrame

Syntax

```
C/C++:    void ggSetVectorChartFrame(GLIMIT *limits);
```

```
F90:      subroutine ggSetVectorChartFrame(limits)
          type(GLIMIT), intent(in) :: limits
```

Arguments

limits.xmin
Horizontal minimum limit of Vector Chart in Graphical coordinates

limits.xmax
Horizontal maximum limit of Vector Chart in Graphical coordinates

limits.ymin
Vertical minimum limit of Vector Chart in Graphical coordinates

limits.ymax
Vertical maximum limit of Vector Chart in Graphical coordinates

Description

The routine ggSetVectorChartFrame() sets up an area in graphical coordinates onto which future Vector Charts are mapped. The graphical coordinate system is set up by the latest calls to the routines ggSetAxesPos() and ggSetAxesScaling().

If ggSetVectorChartFrame() is not called or if the routine ggRestoreVectorSettings() has been called the default area is used. That is the area represented by the intersection of the limits of the horizontal (X) and vertical (Y) axes.

See Also

Page 122
ggSetAxesPos
ggSetAxesScaling
ggAddVectors
ggRestoreVectorSettings

ggSetVectorLimits

Syntax

```
C/C++:    void ggSetVectorLimits(float smin, float smax);
```

```
F90:      subroutine ggSetVectorLimits(smin, smax)
          real, intent(in) :: smin, smax
```

Arguments

smin
Minimum absolute strength of vector that may be displayed using ggAddVectors()

smax
Maximum absolute strength of vector that may be displayed using ggAddVectors()

Description

The routine ggSetVectorLimits() defines the minimum and maximum absolute vector strength that may be drawn by subsequent calls to ggAddVectors(). Any absolute value of vector strength outside the range of **smin** to **smax** are not drawn.

If **smin** > **smax** the two values are inter-changed.

If **smin** and **smax** are both set to equal values then clipping is switched off and all vectors with strength not equal to zero are drawn. The routine ggRestoreVectorSettings() also switches clipping off.

See Also

Page 122
ggAddVectors
ggRestoreVectorSettings

ggTransformGraphPoint

Syntax

```
C/C++:      void ggTransformGraphPoint(float xgr, float ygr, GPOINT *point);
```

```
F90:        subroutine ggTransformGraphPoint(xgr, ygr, point)

            float, intent(in) :: xgr, ygr
            type(GPOINT), intent(out) :: point
```

Arguments

xgr
Value giving the X part of the graphical axes coordinates of a point

ygr
Value giving the Y part of the graphical axes coordinates of a point

point.x
Returns a real value giving the X part of the user space coordinates of a point, equivalent to **xgr**

point.y
Returns a real value giving the Y part of the user space coordinates of a point, equivalent to **ygr**

Description

The routine ggTransformGraphPoint() converts the graphical coordinates of a specified point (**xgr**, **ygr**), defined with respect to the graphical axes system set up by the last axis definition calls or by one of the axis-dependent Basic Complete Routines, to user space coordinates returned as **point.x** and **point.y**.

See Also

Page 166
ggTransformSpacePoint

ggTransformSpacePoint

Syntax

```
C/C++:      void ggTransformSpacePoint(float xsp, float ysp, GPOINT *point);
```

```
F90:        subroutine ggTransformSpacePoint(xsp, ysp, point)

            real, intent(in) :: xsp, ysp
            type(GPOINT), intent(out) :: point
```

Arguments

xsp
Value giving the X part of the user space coordinates of a point

ysp
Value giving the Y part of the user space coordinates of a point

point.x
Returns a real value giving the X part of the graphical axes coordinates of a point, equivalent to **xsp**

point.y
Returns a real value giving the Y part of the graphical axes coordinates of a point, equivalent to **ysp**

Description

The routine ggTransformSpacePoint() converts the user space coordinates of a specified point (**xsp**, **ysp**) to graphical axes coordinates returned as **point.x**, **point.y**, defined with respect to the graphical axes system set up by ggSetAxesPos() and ggSetAxesScaling() or by one of the complete graph or chart routines.

See Also

Page 167
ggSetAxesPos
ggSetAxesScaling
ggTransformGraphPoint

Appendix A

DEFAULTS

Default Parameters

The GINOGRAF default parameters are as follows:

Axes Definition Defaults

Axis position and size

The axis positioning and size is calculated in relation to the available drawing area or current window.

Axis scaling

scale = GLINEARTYPE3 (linear scaling with precisely **nints** intervals, **vbeg** on the first interval and **vend** on the last interval).

nints = 9
vbeg = 1.0
vend = 10.0

Axis annotation

Number of decimal places - Up to 2
Scale power factor = None
Scale type display = *10 to the power n (where -2>n>2)

Annotation attributes

Numerical and string annotation is displayed at each major tick mark starting at the first of each axes. Items on the X axis are displayed centrally justified in the horizontal direction above or below the tick mark. Items on the Y axis are displayed bottom justified in the vertical direction either to the left or right of the axis. All items are displayed horizontally (at 0 degrees) at the current character size.

Data Value Display Defaults

Value position

Values are displayed centrally over coordinate position for ggAddGraphValues() or at centre of column or area for ggAddHistogramValues(), ggAddBarChartValues(), ggAddStepChartValues() or ggAddAreaChartValues().

Value orientation

Values are centre justified and displayed horizontally (at 0 degrees) in the current character size.

Prefix, suffix strings

Null

Chart Defaults

Histogram and bar chart axis

The X axis is the default base axis for a bar chart or histogram where neither or both axes have been defined as discrete axes.

Block Filling Attributes

Colour index offset = 20
Azimuth angle = 30 degrees
Elevation angle = 30 degrees
Depth fraction = 1.0
Top face lightness factor = 0.67
Side face lightness factor = 0.33

Vector Chart Defaults

Mapping

Vector Charts are mapped onto the area represented by intersection of current horizontal (X) and vertical (Y) axes.

Clipping

All vectors with strength not equal to zero are displayed.

Scaling

Maximum vector strength is represented by an arrow of one unit square length.

Polar Chart Defaults

Polar axis parameters

Polar axes are placed in the centre of the drawing area or current window with the radius being 0.375 of the minimum width or height. Linear scaling type GLINEARTYPE3 is the default for the R axis.

Pie Chart Defaults

Pie chart size & position

The pie chart size and position is calculated in terms of the available drawing area or current window with the radius being 0.375 of the minimum width or height.

Pie chart start angle

The default start angle for a pie chart is 0.0 degrees (three o’clock).

Pie chart annotation

The default annotation type for the complete pie chart routines is to write the segment labels horizontally within the segment (unless they don’t fit) within a masked box.

Pie chart boundary

On

Text Chart Defaults

Text chart attributes

The default dimensions of a text chart column is 10.0 * current character width by twice the current character height times the number of rows, with no header cell. The entries are centre justified. The column frame is drawn in the current GINO line style.

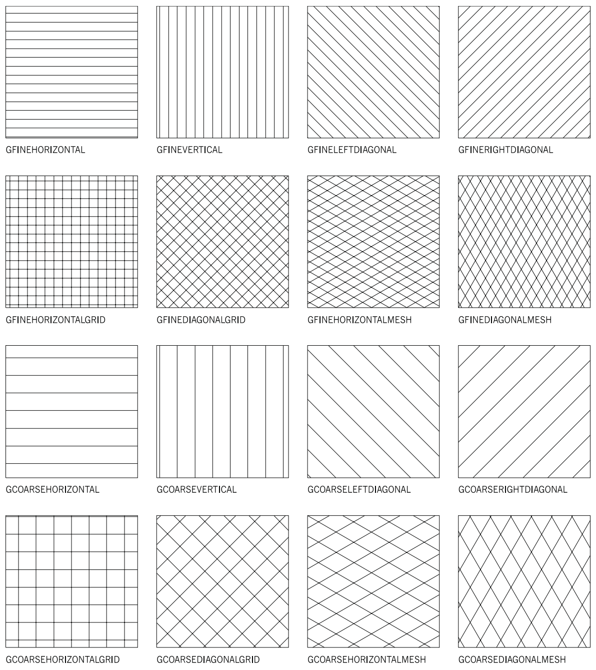
Drawing Attributes

Area fill and hatch styles

The area fill and hatch styles are selected with the variable **fill** in GINOGRAF filling routines. The possible values of **fill** are listed below.

1	`GFINEHORIZONTAL`	fine horizontal hatch
2	`GFINEVERTICAL`	fine vertical hatch
3	`GFINELEFTDIAGONAL`	fine left diagonal hatch
4	`GFINERIGHTDIAGONAL`	fine right diagonal hatch
5	`GFINEHORIZONTALGRID`	fine horizontal grid
6	`GFINEDIAGONALGRID`	fine diagonal grid
7	`GFINEHORIZONTALMESH`	fine horizontal mesh
8	`GFINEDIAGONALMESH`	fine vertical mesh
9	`GCOARSEHORIZONTAL`	coarse horizontal hatch
10	`GCOARSEVERTICAL`	coarse diagonal hatch
11	`GCOARSELEFTDIAGONAL`	coarse left diagonal hatch
12	`GCOARSERIGHTDIAGONAL`	coarse right diagonal hatch
13	`GCOARSEHORIZONTALGRID`	coarse horizontal grid
14	`GCOARSEDIAGONALGRID`	coarse diagonal grid
15	`GCOARSEHORIZONTALMESH`	coarse horizontal mesh
16	`GCOARSEDIAGONALMESH`	coarse diagonal mesh

Hatch styles can be redefined with the GINO routine gDefineHatchStyle().



Default Hatch Styles

Line styles

The line styles are selected with the variable **line** in GINOGRAF filling routines. Default line styles are solid, with 0.2mm width (or the device default), and have the colour index equal to the line style index.

Colours 0 to 10 are defined by GINO as:

0	`GBACKGROUND`
1	`GBLACK`
2	`GRED`
3	`GORANGE`
4	`GYELLOW`
5	`GGREEN`
6	`GCYAN`
7	`GBLUE`
8	`GMAGENTA`
9	`GBROWN`
10	`GWHITE`

Some devices may support fewer colours than the standard GINO range.

Symbols

The standard GINO symbols are as follows:

GSPOT	GUP	GDOWN	GPLUS	GCROSS	GBOX	GDIAMOND	GCIRCLE	GSTAR
●	△	▽	+	×	□	◇	○	✳

Appendix B

ERROR AND WARNING MESSAGES

GINOGRAF Errors and Warnings

1	**Number of columns negative or zero** The number of columns in a barchart or histogram cannot be negative or zero
2	**Value for log axis is negative or zero** Log axes in barchart or histogram cannot have a value of zero
3	**Discrete axis not specified** Either the X axis or Y axis needs to be set to ‘discrete’ (perpendicular to bars) when defining a barchart
4	**Error in log data** A log axis has been defined with a barchart or histogram and incorrect data has been passed to it
5	**Number of pie chart segments outside range 1-50** ggPlotPieChart() must have a number of segments in the range 1-50
6	**All values are zero** A piechart or percentage column has been attempted with all data values being set to zero
7	**Annotation number of decimal places out of range** ndp in ggSetAxesAnnotation() must be in the range -9 to 9
8	**Negative line style** A line style has been requested with a negative value

9 **Illegal or incompatible annotation scale type**
Certain scale types require that npower is a multiple of 3

10 **Drawing area available to GINOGRAF is zero**
The Window or Paper size is too small. Use gSetDrawingLimits() or gWindow2D() to increase the area

11 **Drawing area width too small for graph/chart**
Increase the available drawing width with ggSetPlotFrame()

12 **Drawing area height too small for graph/chart**
Increase the available drawing height with ggSetPlotFrame()

13 **X value negative or zero for log axis**
Self explanatory

14 **Y value negative or zero for log axis**
Self explanatory

15 **Number of points to be plotted less than two**
All GINOGRAF point-plotting routines require at least two points

16 **Number os symbols to be plotted negative or zero**
ggAddGraphMarkers() routine requires at least 1 symbol to be plotted

17 **Axis length negative or zero**
Self explanatory

18 **Axis scaling type out of range**
Axis scaling type must be set to one of the predefined types

19 **Number of intervals negative or zero**
The number of intervals defined for an axis must be 1 or greater

20 **Range of data values zero**
Data values do not have any range

21 **Negative data range**
Cannot have negative data on a polar axis

22 **Number of list elements outside range 1-50**
ggSetPieChartExplosion() must have a number of elements in the range 1-50

23	**A list element is outside range 1-50** ggSetPieChartExplosion() has been called with element number outside allowable range
24	**Negative explosion factor requested** ggSetPieChartExplosion() cannot have a negative explosion factor
25	**Available polygon workspace is low when using area fill** More polygon workspace is required - Increase the value in gDefinePolygonWorkspace()
26	**Negative number of unmarked points requested** ggAddGraphMarkers() must have a positive value for unmarked points
27	**Symbol number out of range** Symbol number must not be negative
28	**Arrowhead type or coordinate type out of range** Check values in ggDrawArrow() or ggAddVectors()
29	**Arrow length is zero** Self explanatory
30	**Pie chart radius is zero** Self explanatory
31	**Negative pie chart radius requested** Self explanatory
32	**Routine used that will not be available in next release** Check Appendix E for list of deprecated routines
33	**Routine no longer available** Check Appendix E for list of deprecated routines
34	**Number of areas negative or zero** Area charts must have a positive number of areas
35	**Number of steps negative or zero** Step Charts must have a positive number of steps
36	**Number of vectors negative or zero** Vector charts must have a positive number of vectors

37 **All vector strength values are zero**
Vector strengths must be positive

38 **Number of error bars negative or zero**
Self explanatory

39 **Number of text boxes negative or zero**
Self explanatory

40 **No characters in graph title**
The string given to ggDrawGraphTitle() is blank

41 **Number of points to be fitted less than two**
ggReturnLineCoeffs() must have more than two points

42 **Number of coefficients required negative or zero**
ggReturnLineCoeffs() must have a positive number of coefficients

43 **Invalid reference line text position**
Check text positions available in routine ggAddReferenceLine()

44 **String length greater than 30 characters - truncated**
Prefix and Suffix strings in ggSetValueTags() cannot be greater than 30 characters

45 **Invalid justification**
Check allowable justification settings in ggSetAxesAttribs(), ggSetTextChartAttribs() or ggSetValueAttribs()

46 **Invalid value position**
Check allowable positions in ggSetValueAttribs()

47 **Invalid breakpoint position**
Check allowable breakpoint positions in ggAddSquareWave()

48 **Invalid line angle**
Check allowable angles for line columns in ggDisplayLineColumn()

49 **Invalid pie chart annotation type**
Check allowable annotation type in ggSetPieChartAnnotation()

50 **Zero extent for graph drawing limits**
Increase sizes in ggSetPlotFrame()

51 **Invalid pie chart annotation box type**
Check allowable box types in ggSetPieChartBoxType()

52 **Invalid value specifier**
Check value specifier

53 **Invalid arrow position**
Check arrow position in ggSetVectorAttribs()

55 **Invalid date format type**
Check date format in ggSetDateFormat()

56 **Invalid date/time increment**
Check date/time increment in ggSetDateAxesScaling()

57 **Incorrect date format**
Check date format in ggSetDateAxesScaling() or ggConvertDates()

58 **Invalid missing value mode**
Check missing value mode in ggDefineMissingValues()

59 **Invalid graph scaling mode**
Check graph scaling mode in ggSetGraphScaling()

60 **Primary dimension of data array less then number of columns**
The specified size of the data array is smaller than the number of columns requested

61 **Gap outside permitted range**
Gap must be in the range 0.0 to 1.0. Negative values are not permitted

62 **Multi-histogram type out of range**
Multi-histogram type must be either GSTACKED or GCLUSTRED

63 **Sub-array limits outside primary dimension of data array**
Sub-array limits must lie within the specified dimensions of the data array

64 **Number of data sets negative or zero**
Number of data sets must be positive

Configuration File Errors

```
*** Incorrect GINOGRAF Serial No. - Program Aborted! ***
```

Check the contents of your configuration file. There should be a serial number line beginning with GRAFSERIAL= followed by a string of ASCII characters. Contact Bradly Associates or your dealer if this error continues.

```
  *** GINOGRAF Evaluation period expired! ***
```

You have a temporary licence which has now expired - Contact Bradly Associates or your dealer.

Appendix C

STRUCTURES

Structures in GINOGRAF

GAREACHART - Data type for area charts

```
typedef struct {
  float s;
  float f;
  float h1;
  float h2;
} GAREACHART;
```

```
type GAREACHART
  sequence
  real :: s
  real :: f
  real :: h1
  real :: h2
end type
```

GBARCHART - Data type for bar charts

```
typedef struct {
  float s;
  float f;
} GBARCHART;
```

```
type GBARCHART
  sequence
  real :: s
  real :: f
end type
```

GERROR - Data type for an error range

```
typedef struct {
  float lower;
  float upper;
} GERROR;
```

```
type GERROR
  sequence
  real :: lower
  real :: upper
end type
```

GSTEPCHART - Data type for step charts

```
typedef struct {
  float s;
  float f;
  float h;
} GSTEPCHART;
```

```
type GSTEPCHART
  sequence
  real :: s
  real :: f
  real :: h
end type
```

GVECTOR - Data type for vector charts

```
typedef struct {
    float direc;
    float stren;
    int   col;
  } GVECTOR;
```

```
type GVECTOR
  sequence
  real    :: direc
  real    :: stren
  integer :: col
end type
```

GINO Structures Used By GINOGRAF

GPOINT - Single 2D coordinate

```
typedef struct {
  float x;
  float y;
} GPOINT;
```

```
type GPOINT
  sequence
  real :: x
  real :: y
end type
```

GLIMIT - 2D coordinate limits

```
typedef struct {
  float xmin;
  float xmax;
  float ymin;
  float ymax;
} GLIMIT
```

```
type GLIMIT
  sequence
  real :: xmin
  real :: xmax
  real :: ymin
  real :: ymax
end type
```

Appendix D

CROSS-REFERENCES

Cross-Reference Summary

GINOGRAF is supplied as either a C library or FORTRAN library. The FORTRAN library includes two bindings; an F77 binding using short names and simple arguments and an F90 binding using long names and structures/optional arguments as appropriate. This Appendix gives the cross-references from both short name to long name and vice-versa.

F77-F90 Cross-Reference

F77 names	**F90 names**
ARECHA	ggPlotAreaChart
AREFI2	ggBlockFillAreaChart
AREFIL	ggFillAreaChart
AREGRA	ggAddAreaChartOutline
AREVAL	ggAddAreaChartValues
ARROW	ggDrawArrow
AXIANN	ggSetAxesAnnotation
AXIATT	ggSetAxesAttribs
AXIDRA	ggDrawAxes
AXIPOS	ggSetAxesPos
AXISCA	ggSetAxesScaling
AXISET	ggRestoreAxesSettings
AXLSTR	ggDrawAxesLabels
AXNENQ	ggEnqAxesAnnotation
AXNSTR	ggDrawAxesTitle
AXPENQ	ggEnqAxesPos

AXSENQ	ggEnqAxesScaling
AXTENQ	ggEnqAxesAttribs
BARCHA	ggPlotBarChart
BARFI2	ggBlockFillBarChart
BARFIL	ggFillBarChart
BARGRA	ggAddBarChartOutline
BARVAL	ggAddBarChartValues
CHT2ST	ggRestoreBlockChartAttribs
CHTATT	ggSetBlockChartAttribs
CHTATQ	ggEnqBlockChartAttribs
COMFI2	ggBlockFillMultiHistogram
COMFIL	ggFillMultiHistogram
CURBEG	ggSetCurveStartConds
CURFIN	ggSetCurveEndConds
DATANN	ggSetDateAxesAnnotation
DATANQ	ggEnqDateAxesAnnotation
DATCON	ggConvertDates
DATFOQ	ggEnqDateFormat
DATFOR	ggSetDateFormat
DATGRA	ggConvertDateToGraph
DATSCA	ggSetDateAxesScaling
DATSCQ	ggEnqDateAxesScaling
DRAPOL	ggDrawPolarAxes
GRAAKI	ggAddAkimaCurve
GRACUR	ggAddGraphCurve
GRAERB	ggAddErrorBars
GRADAT	ggConvertGraphToDate
GRAFI2	ggFillBetweenDatasets
GRAFIL	ggFillBelowDataset
GRALIN	ggAddGraphLine
GRAMOV	ggMoveToGraphPoint
GRAPH	ggPlotGraph
GRAPOL	ggAddGraphPolyline
GRAPOP	ggAddPopulationGraph
GRASCA	ggSetGraphScaling
GRASPA	ggTransformGraphPoint
GRASPL	ggAddGraphSpline
GRASQU	ggAddSquareWave
GRASYM	ggAddGraphMarkers
GRAVAL	ggAddGraphValues

GRBENQ	ggEnqPlotFrame
GRDSYM	ggSetGridMarker
GRDSYQ	ggEnqGridMarker
GRFBND	ggSetPlotFrame
GRFMOD	ggSetGraphCharMode
GRFTTL	ggDrawGraphTitle
GRID	ggAddGrid
HISCHA	ggPlotHistogram
HISFI2	ggBlockFillHistogram
HISFIL	ggFillHistogram
HISGRA	ggAddHistogramOutline
HISVAL	ggAddHistogramValues
LINFIT	ggReturnLineCoeffs
PIAENQ	ggEnqPieChartAnnotation
PIEANG	ggSetPieChartStartAngle
PIEANN	ggSetPieChartAnnotation
PIEBOX	ggSetPieChartBoxType
PIEBSW	ggSetPieChartBoundSwitch
PIECHA	ggPlotPieChart
PIEENQ	ggEnqPieChartSettings
PIEEXP	ggSetPieChartExplosion
PIEPAP	ggSetPieChartFrame
PIESET	ggRestorePieChartSettings
POLXY	ggPlotXYPolarChart
REFLIN	ggAddReferenceLine
SEGCHA	ggAddPieChartSegment
SETPOL	ggSetPolarChartAttribs
SPAGRA	ggTransformSpacePoint
STPCHA	ggPlotStepChart
STPFI2	ggBlockFillStepChart
STPFIL	ggFillStepChart
STPGRA	ggAddStepChartOutline
STPVAL	ggAddStepChartValues
TDFENQ	ggEnqTextChartAttribs
TEXDEF	ggSetTextChartAttribs
TEXFIL	ggDisplayFillColumn
TEXGEN	ggDisplayGeneratedColumn
TEXLIN	ggDisplayLineColumn
TEXPER	ggDisplayPercentageColumn
TEXSTR	ggDisplayStringColumn

TEXSYM	ggDisplayMarkerColumn
TEXVAL	ggDisplayValueColumn
VALATT	ggSetValueAttribs
VALMIS	ggDefineMissingValues
VALTXT	ggSetValueTags
VATENQ	ggEnqValueAttribs
VECBND	ggSetVectorChartFrame
VECBNQ	ggEnqVectorChartFrame
VECCLP	ggSetVectorLimits
VECCLQ	ggEnqVectorLimits
VECGRA	ggAddVectors
VECSCA	ggSetVectorAttribs
VECSCQ	ggEnqVectorAttribs
VECSET	ggRestoreVectorSettings

F90-F77 Cross-Reference

F90 names	**F77 names**
ggAddAkimaCurve	GRAAKI
ggAddAreaChartOutline	AREGRA
ggAddAreaChartValues	AREVAL
ggAddBarChartOutline	BARGRA
ggAddBarChartValues	BARVAL
ggAddErrorBars	GRAERB
ggAddGraphCurve	GRACUR
ggAddGraphLine	GRALIN
ggAddGraphPolyline	GRAPOL
ggAddGraphSpline	GRASPL
ggAddGraphMarkers	GRASYM
ggAddGraphValues	GRAVAL
ggAddGrid	GRID
ggAddHistogramOutline	HISGRA
ggAddHistogramValues	HISVAL
ggAddPieChartSegment	SEGCHA
ggAddPopulationGraph	GRAPOP
ggAddReferenceLine	REFLIN
ggAddSquareWave	GRASQU
ggAddStepChartOutline	STPGRA
ggAddStepChartValues	STPVAL
ggAddVectors	VECGRA

ggBlockFillAreaChart	AREFI2
ggBlockFillBarChart	BARFI2
ggBlockFillHistogram	HISFI2
ggBlockFillMultiHistogram	COMFI2
ggBlockFillStepChart	STPFI2
ggConvertDates	DATCON
ggConvertDateToGraph	DATGRA
ggConvertGraphToDate	GRADAT
ggDefineMissingValues	VALMIS
ggDisplayFillColumn	TEXFIL
ggDisplayGeneratedColumn	TEXGEN
ggDisplayLineColumn	TEXLIN
ggDisplayPercentageColumn	TEXPER
ggDisplayStringColumn	TEXSTR
ggDisplayMarkerColumn	TEXSYM
ggDisplayValueColumn	TEXVAL
ggDrawArrow	ARROW
ggDrawAxes	AXIDRA
ggDrawAxesLabels	AXLSTR
ggDrawAxesTitle	AXNSTR
ggDrawGraphTitle	GRFTTL
ggDrawPolarAxes	DRAPOL
ggEnqAxesAnnotation	AXNENQ
ggEnqAxesAttribs	AXTENQ
ggEnqAxesPos	AXPENQ
ggEnqAxesScaling	AXSENQ
ggEnqBlockChartAttribs	CHTATQ
ggEnqDateAxesAnnotation	DATANQ
ggEnqDateAxesScaling	DATSCQ
ggEnqDateFormat	DATFOQ
ggEnqGridMarker	GRDSYQ
ggEnqPieChartAnnotation	PIAENQ
ggEnqPieChartSettings	PIEENQ
ggEnqPlotFrame	GRBENQ
ggEnqTextChartAttribs	TDFENQ
ggEnqValueAttribs	VATENQ
ggEnqVectorAttribs	VECSCQ
ggEnqVectorChartFrame	VECBNQ
ggEnqVectorLimits	VECCLQ
ggFillAreaChart	AREFIL

ggFillBarChart	BARFIL
ggFillBelowDataset	GRAFIL
ggFillBetweenDatasets	GRAFI2
ggFillHistogram	HISFIL
ggFillMultiHistogram	COMFIL
ggFillStepChart	STPFIL
ggMoveToGraphPoint	GRAMOV
ggPlotAreaChart	ARECHA
ggPlotBarChart	BARCHA
ggPlotGraph	GRAPH
ggPlotHistogram	HISCHA
ggPlotPieChart	PIECHA
ggPlotStepChart	STPCHA
ggPlotXYPolarChart	POLXY
ggRestoreAxesSettings	AXISET
ggRestoreBlockChartAttribs	CHT2ST
ggRestorePieChartSettings	PIESET
ggRestoreVectorSettings	VECSET
ggReturnLineCoeffs	LINFIT
ggSetAxesAnnotation	AXIANN
ggSetAxesAttribs	AXIATT
ggSetAxesPos	AXIPOS
ggSetAxesScaling	AXISCA
ggSetBlockChartAttribs	CHTATT
ggSetCurveEndConds	CURFIN
ggSetCurveStartConds	CURBEG
ggSetDateAxesAnnotation	DATANN
ggSetDateAxesScaling	DATSCA
ggSetDateFormat	DATFOR
ggSetGraphCharMode	GRFMOD
ggSetGraphScaling	GRASCA
ggSetGridMarker	GRDSYM
ggSetPieChartAnnotation	PIEANN
ggSetPieChartBoundSwitch	PIEBSW
ggSetPieChartBoxType	PIEBOX
ggSetPieChartExplosion	PIEEXP
ggSetPieChartFrame	PIEPAP
ggSetPieChartStartAngle	PIEANG
ggSetPlotFrame	GRFBND
ggSetPolarChartAttribs	SETPOL

ggSetTextChartAttribs	TEXDEF
ggSetValueAttribs	VALATT
ggSetValueTags	VALTXT
ggSetVectorChartFrame	VECBND
ggSetVectorAttribs	VECSCA
ggSetVectorLimits	VECCLP
ggTransformGraphPoint	GRASPA
ggTransformSpacePoint	SPAGRA

Appendix E

DEPRECATED ROUTINES

Deprecation Procedure

This appendix contains routines that are being deprecated due to the developing nature of GINOGRAF as it keeps in line with changes in the graphics and general computing environment.

A routine will go through two intermediate stages prior to being removed from the GINOGRAF library:

Stage 0:

Routine has no further use in GINOGRAF library. Documentation will be removed from the reference chapter and temporarily placed in this Appendix. The routine will not however be removed from the library.

Stage 1:

Routine will generate a warning message but will run correctly. Documentation will be removed from the reference chapter and placed in this Appendix.

Stage 2:

Routine will generate an error message and will not have any effect on a users program. Its routine specification and arguments will remain in this Appendix, but without the description. Alternative routines (where applicable) will be indicated.

Stage 3:

The routine will be removed from the GINOGRAF library.

Each stage represents one major release of GINOGRAF which gives about 2-3 years in order to facilitate changes to an application program to reflect any deprecations of a routine.

However, it is stressed that no routine will be deprecated without alternatives being provided or where it is offering a facility that has fallen out of use, and in both cases discussed by the GINO Technical Committee. Any problems that are encountered due to routine deprecation should be addressed to the Product Development Manager of **Bradly Associates Limited**.

There are currently NO routines at any level of deprecation in this version of GINOGRAF.

Due to a rationalisation, the following routines are being phased out of the GINOGRAF library due to duplication or lack of use. The following table lists the F77 routine name and the alternative routine that should be called if appropriate.

Fortran-77	**New routine if appropriate**
`PIEFIL`	`ggPlotPieChart`
`PIESTR`	`ggPlotPieChart`
`SEGFIL`	`ggAddPieChartSegment`
`SEGSTR`	`ggAddPieChartSegment`

Index