dGE Version 2

USER GUIDE

NOTICE

The information contained in this document is subject to change without notice. While every effort is made ensure the accuracy of the information contained herein, Bits Per Second shall not be liable for any errors or for any consequential damages in connection with the performance or use of this material.

This document contains proprietary information which is protected by copyright. No part of this document may be photocopied, reproduced or translated without the written consent of Bits Per Second Ltd.

COPYRIGHT © 1984,88 by Bits Per Second Ltd,

dGE is a trademark of Bits Per Second Ltd.
Clipper is a trademark of Nantucket Inc.
dBASE is a trademark of Ashton-Tate
FoxBASE+ is a trademark of FOX International
Laserjet is a product of Hewlett-Packard Co
IBM PC and IBM PS/2 are products of International Business Machines Corp.
IBM is a registered trademark International Business Machines Corp.

PRINTING HISTORY First edition February 1988

dGE concept and development Graeme Knott
User Guide Kim Jones

All illustrations in this guide are reproduced directly from dGE screens, using a HP Laserjet series II printer and QuadEGA card in 640 x 350 Enhanced mode.

About this Guide

This is the User Guide for *dGE Version 2*, the database Graphics Extension for *dBASE* and *C* languages.

Certain fundamental features of *dGE Version 1* have been altered, in particular the increase in co-ordinate space to 1000 x 1350 which has been introduced in response to the improved video standards of current IBM machines.

The guide does not cover Version 1 operation. However, for those who have already developed programs for Version 1 there is a "mode switch" which will enable you to continue development of these programs while taking advantage of Version 2 features.

It is assumed that readers have some familiarity with the host language (*dBASE* or *C*). The syntax of *dGE* is consistent between *Clipper*, *FoxBASE+* and *C*, but differs in *dBASE III*. All programming examples in this guide are written in the former style. We anticipate that *dBASE IV* will come into line with this during 1988.

You are strongly recommended to examine the source code of the numerous example programs included on the issue disks, in order to gain a further insight into programming with *dGE*.

Table Of Contents

Chapter 12 – dGE Function Reference 12-1

Appendix A

Quick reference guide to functions

Functions which return a value are shown as –

Value = Function(..)

where

Value = N	numeric value
Value = C	character value

bargraph(*X, Y, Inc, Mode, Group*)
datastore(*Amp, Patt, Sign, Colour*)

bestfit(*Wid, Ht, Lstyle, Lcolour*)

boxfill(*X, Y, Wid, Ht, Patt, Colour*)

clrline(*X, Y, Nchars*)

clrscreen()

clrstring()

clrwin(*X0, Y0, X1, Y1*)

datapc(*Pc*)

datarange(*First, Last*)

datareset()

datastore(*P1, P2, P3, P4*)

diskfile(*Mode, File*)

drawcircle(*X, Y, Rad, Ang1, Ang2, Mode, Style, Colour*)

drawline(*X0, Y0, X1, Y1, Lmode, Lstyle, Lcolour*)

drawvec(*Len, Ang, Update, Lmode, Lstyle, Lcolour*)

drawxy(*Xrel, Yrel, Update, Lmode, Lstyle, Lcolour*)

C = edstring(*X, Y, Cset, Colour, String*)

fixpos(*X, Y*)

N = getasin(*Value (x10000), SinCos*)

N = getcc()

C = getchar(*Mode*)

N = getcurx()

N = getcury()

N = getmax()

N = getmean()

N = getmin()

N = getsd()

N = getsin(*Ang, SinCos*)

C = getstring(*X, Y, Cset, Nchars, Colour*)

hlcgraph(*X0, Y0, Xinc, Colour*)
datastore(*A, B, C, Xpos*)

labelpie(*Xoff, Rad, LabLen, Cset, Mode, Colour, String*)

labelx(*X, Y, Xinc, LabLen, Cset, Mode, Colour, String*)

labely(*X, Y, Yinc, LabLen, Cset, Mode, Colour, String*)

loadcset(*Cset, File*)

minmax(*Wid, Lstyle, Lcolour*)

movevec(*Len, Ang*)

movexy(*Xrel, Yrel*)

piechart(*X, Y, Rad*)
datastore(*Value, Patt, Expl, Colour*)

plotcset(*DgeCset, Height, Width, Font*)

plotoff()

ploton(*Hoff, Voff, Xlen, Mode, Units*)

plotpen(*DgeColour, PenNo, PenThick*)

polaraxes(*X, Y, Rad, Ndivs, Colour*)

polargraph(*X, Y, Cycles, Style, Colour*)
datastore(*Amp, Icon, Ang, 0*)

polyline(*X, Y, Lmode, Lstyle, Lcolour*)
datastore(*Xrel, Yrel, 0, 0*)

polyvec(*Pos1, Pos2, Mode, Ang, Lmode, Lstyle, Lcolour*)
datastore(*A, B, Move, 0*)

printfile(*Mode, File*)

printpcl(*Mode, Hoffset, Voffset, Density*)

printscrn()

sayicon(*X, Y, Mode, IconId, Colour*)

saystring(*X, Y, Cset, Mode, Colour, String*)

setdelim(*AscChar*)

sethires(*Mode*)

setpal(*Bg, Int, Fg*)

settext()

setver(*Version*)

setwin(*X0, Y0, X1, Y1*)

shade(*X, Y, Pattern, Colour*)

stats(*Wid, Mode, Lstyle, Lcolour*)

timedata(*Amp1, Amp2, Amp3, Amp4*)

timegraph(X, Y, Xint, Xpts, Ch1, Ch2, Ch3, Ch4)
timedata(Amp1, Amp2, Amp3, Amp4)

vecicon(Len, Ang, Mode, IconId, Colour)

vecstring(Len, Ang, Cset, Mode, Colour, String)

xyaxes(X, Y, Xlen, Ylen, Xdivs, Ydivs, Mode, Colour)

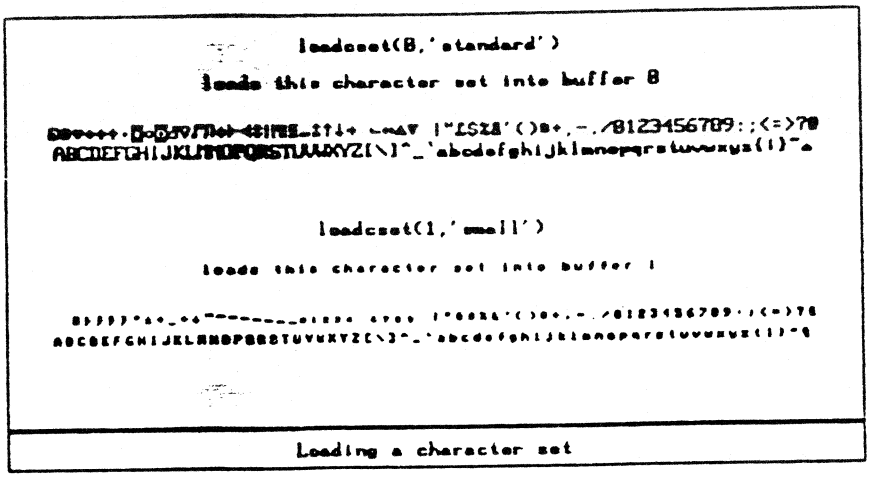
xygraph(X, Y, Xinc, Mode, Colour)
datastore(Yamp, Icon, Xpos, 0)

Appendix B

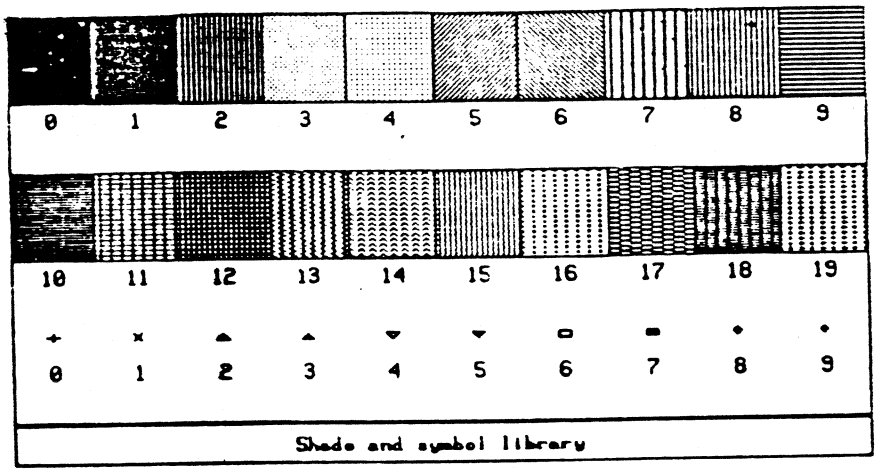
Graphics layout grid

Appendix C

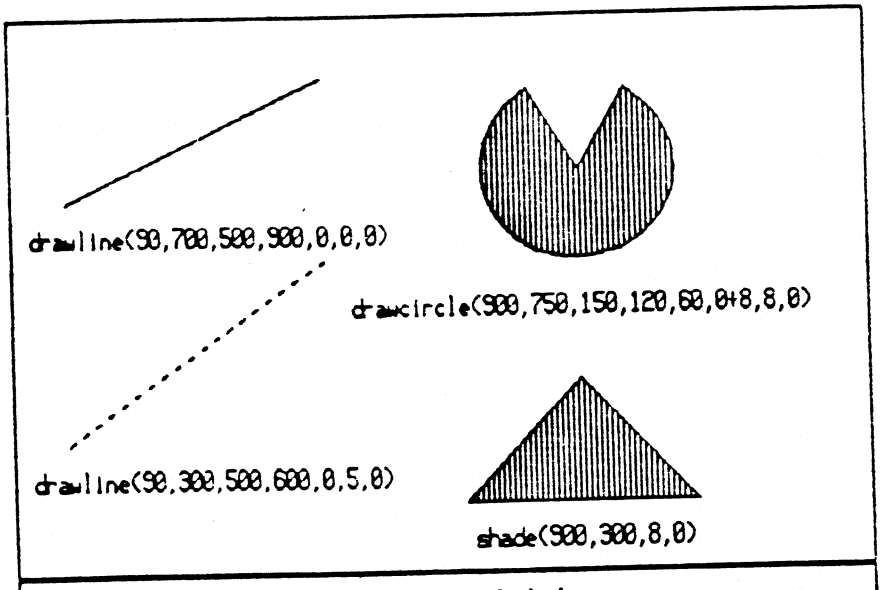
dGE Sample Screens



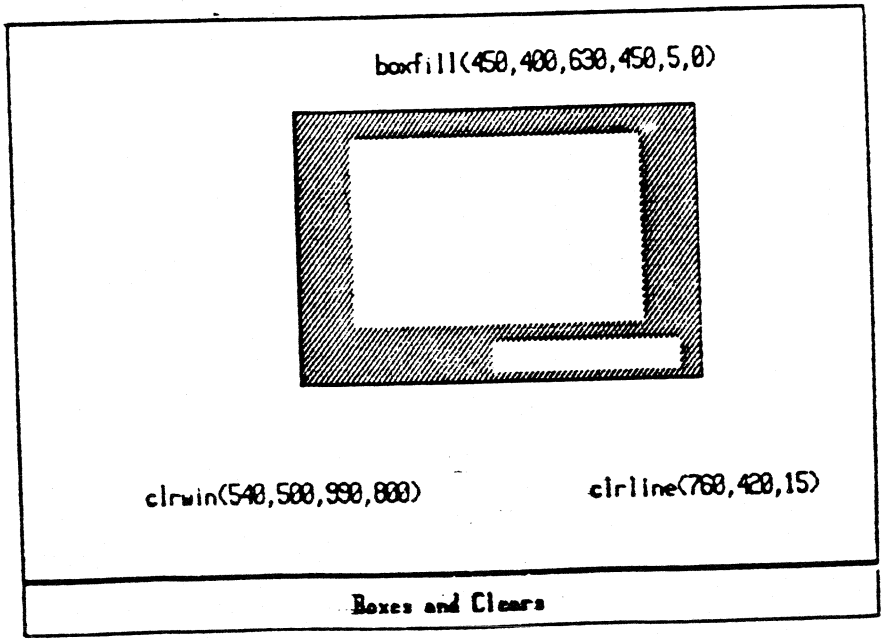
Loading a character set



Shade and symbol library



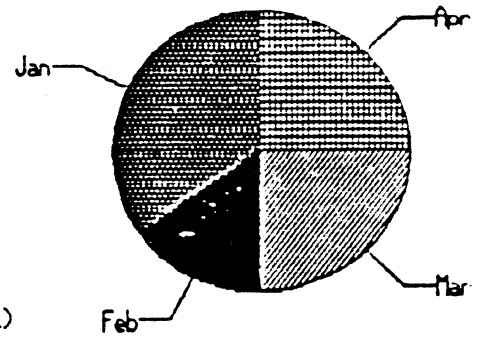
Line, circle and shade



Boxes and Clears

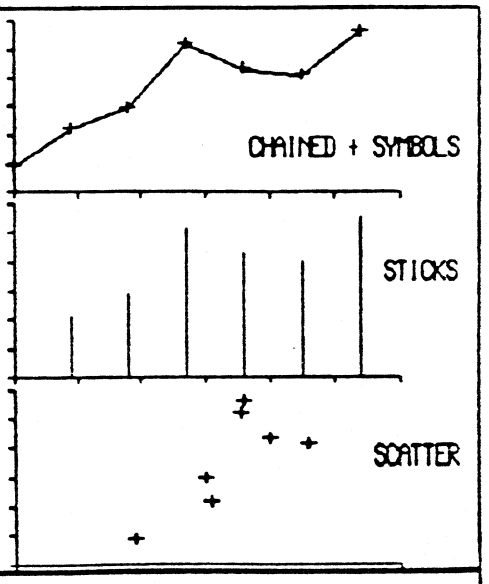
Notes:

```
datareset()
datastore(70, 1,0,0)
datastore(30, 0,1,1)
datastore(50, 5,0,0)
datastore(50,12,0,0)
piechart(300,600,230)

st = 'JanFebMarApr'
labelpie(50,310,3,1,0,0,st)
```

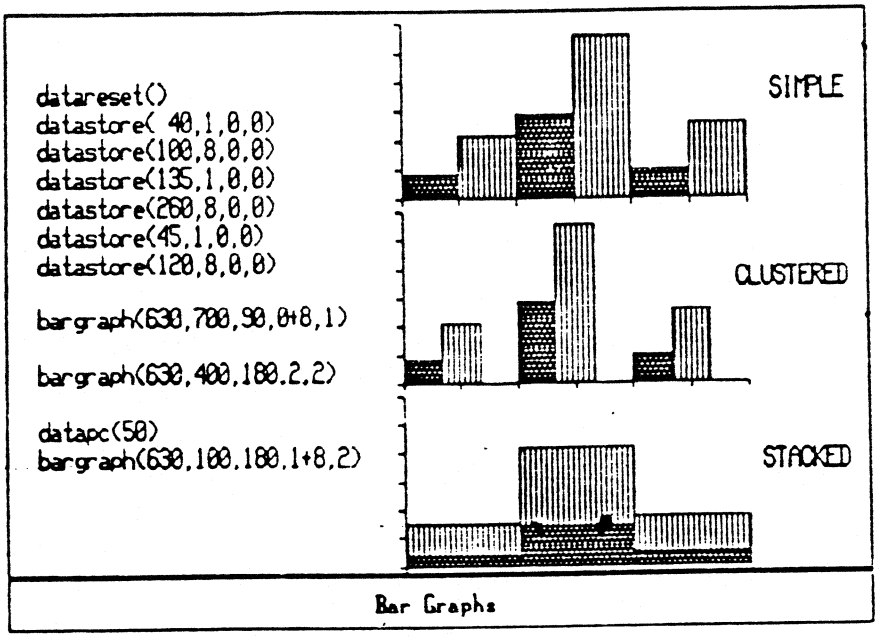


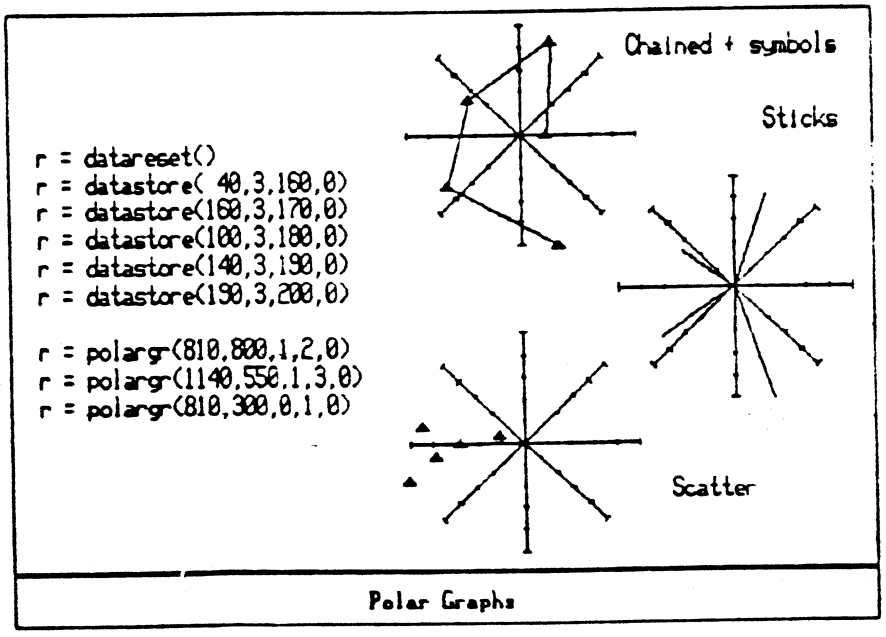
Pie Chart

```
datareset()
datastore( 40,0,190,0)
datastore(100,0,310,0)
datastore(135,0,300,0)
datastore(240,0,355,0)
datastore(300,0,395,0)
datastore(190,0,460,0)
datastore(260,0,360,0)

xygraph(630,100,0,1,0)

xygraph(630,400,90,3,0)

xygraph(630,700,90,2,0)
```

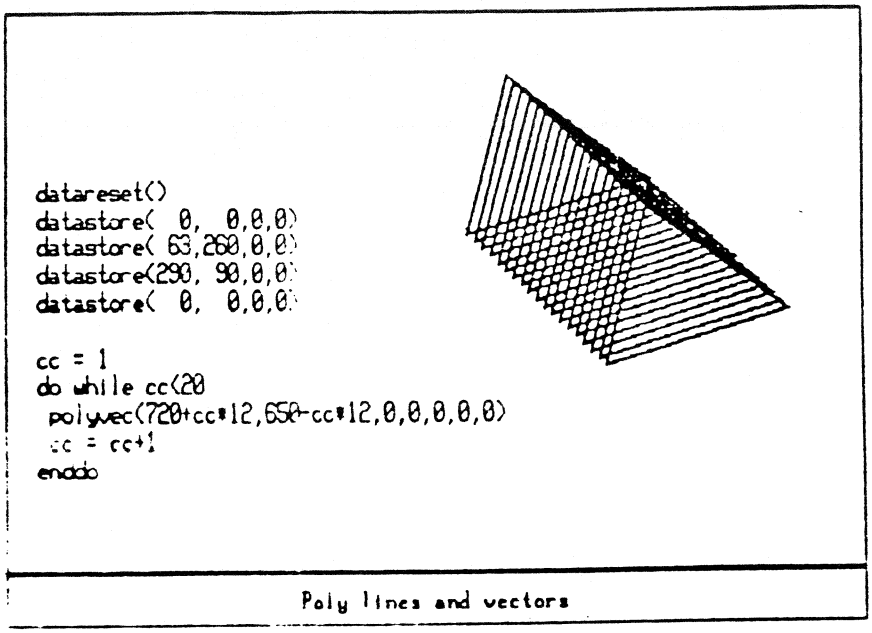


X-Y graphs


datareset()
datastore(40,1,0,0)
datastore(100,8,0,0)
datastore(135,1,0,0)
datastore(260,8,0,0)
datastore(45,1,0,0)
datastore(120,8,0,0)
bargraph(630,700,90,0+8,1)
bargraph(630,400,100,2,2)
datapc(50)
bargraph(630,100,100,1+8,2)
SIMPLE
CLUSTERED
STACKED
Bar Graphs


r = datareset()
r = datastore(40,3,160,0)
r = datastore(150,3,170,0)
r = datastore(100,3,180,0)
r = datastore(140,3,190,0)
r = datastore(190,3,200,0)
r = polargr(810,800,1,2,0)
r = polargr(1140,550,1,3,0)
r = polargr(810,300,0,1,0)
Chained + symbols
Sticks
Scatter
Polar Graphs

Notes:



Poly lines and vectors

Notes:

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

R

S

T

U

V

W

X

Y

Chapter 1

An introduction to dGE

The history of dGE

When *dGE* was developed in 1984 it provided the first practical tool to enable users of *dBASE II* to create graphs and charts from within a *dBASE* program.

Its principle was simple: a library of subroutines was loaded into the upper part of the *dBASE II* code segment from where it could be "called" from within an executing *dBASE* program.

It was able to produce graphics by issuing the appropriate operating system call to change the mode of the screen into one which supported an *all-points-addressable* mode of access. Thereafter the *dGE* functions were able to draw figures in response to function calls within the user's program.

Graphics and text could be mixed on screen, and the commands used to generate them could, in turn, be mixed freely in *dBASE* programs with standard *dBASE* commands.

Basic graphics facilities included -

- line and circle drawing
- drawing of icons
- shading areas
- clearing windows
- drawing axes for graphs

Utilities included printing the screen on a printer, reading and writing a graphics screen to a disk file, loading different character sets and calculating trigonometric functions.

Its most powerful feature was presenting data in pictorial form. Using a minimum of commands you could present a set of data as -

- a pie chart,
- a bar graph (simple, stacked or clustered),
- a cartesian graph
- a polar graph
- a polyline (a diagram defined by a list of co-ordinate points)

Additionally, mean, standard deviation and best-fit lines could be generated automatically and superimposed on a graph.

In following years *dGE* was extended to support a variety of video standards on a variety of microcomputers, and was adapted to work with *dBASE III*, *C*, *BASIC* and several *dBASE* clones and compilers.

dGE Version 2 has been developed in the light of four year's experience and in response to the numerous requests for new features made by users of *dGE Version 1*.

dGE Version 2 includes several fundamental changes, in particular the change in the co-ordinate space from 150 by 225 to 1000 by 1350.

All the features of *dGE Version 1* may be accessed by switching to a *compatibility* mode, which allows *dGE Version 1* programs to be executed without alteration.

How does dGE work ?

dGE controls the high resolution screen through the graphics kernel SETDGE, which must be loaded into memory before running your program. Once loaded into memory SETDGE remains there until the end of the session. The program is the same whether you are running *dBASE*, *Clipper*, *FoxBASE+* or *C*.

Within a program, *dGE* commands are included in the form of single line statements which invoke functions, generally qualified by a list of parameters.

These functions are defined in *linkage* modules which are specific to the host program or language you are working with. *dGE* functions communicate from the linkage module to the graphics kernel SETDGE. Control is momentarily transferred outside the host program, where the graphics function is executed before returning to the next statement in your code.

dGE commands can be placed anywhere in a program. Typically your first command will be to switch the screen into graphics mode, after which you can start issuing commands to draw figures or write text.

Chapter 2

Installing dGE

Files on the issue disks

dGE is supplied on either two 5.25 inch disks or one 3.5 inch disk. Due to its greater capacity, the 3.5 inch disk contains all of the files and directories on the same disk.

Disk 1 - Drivers and utilities

\README.1ST	Latest information on *dGE*
\DINSTALL.BAT	Driver installation batch file
\BUGS.DOC	Latest bug report
\DRIVERS\EGA\SETDGE.COM	for EGA
\DRIVERS\CGM\SETDGE.COM	for CGA mono
\DRIVERS\CGC\SETDGE.COM	for CGA colour
\DRIVERS\HRC\SETDGE.COM	for HERCULES mono
\CSETS\EGAHRC*.CHR	character files EGA and HERCULES
\CSETS\CGA *.CHR	character files CGA
\UTILS\VIEW.COM	view utility
\UTILS\CONFIGP.EXE	matrix printer configurator
\UTILS\HGC.COM	HERCULES mode switch

Disk 2 - Linkage and demonstration programs

\HINSTALL.BAT	installation batch file
\LINKAGE\DB3\DGE.PRG	linkage for *dBASE III*
\LINKAGE\DB3\CONFIG.DB	*dBASE* config file
\LINKAGE\CLIP\DGEFUNS.PRG	linkage for *Clipper*
\LINKAGE\CLIP\DGECLASM.OBJ	second link module
\LINKAGE\CLIP\CLIPCOMP.BAT	compile and link batch file
\LINKAGE\FOX\DGEUDFS.PRG	linkage for *FoxBASE+*
\LINKAGE\FOX\DGE.BIN	second link module
\LINKAGE\C\DGEC.C	linkage for *C*
\LINKAGE\C\DGECASM.ASM	second link module
\LINKAGE\C\DGEC.H	*C* header file
\LINKAGE\C\MSCCOMP.BAT	compile and link batch file
\DEMOPROG\DGEDEFS.PRG	*dGE* public definitions for *dBASE*
\DEMOPROG*.DBF	database files
\DEMOPROG\DB3*.PRG	*dBASE* demo programs
\DEMOPROG\DB3*.BAT	*dBASE* demo batch files
\DEMOPROG\FOXCLIP*.PRG	*FoxBASE+* demo programs
\DEMOPROG\FOXCLIP*.BAT	*FoxBASE+* demo batch files
\DEMOPROG\C*.C	*C* source files
\DEMOPROG\C*.H	*C* header file
\DEMOPROG\C*.DAT	*C* data files
\DEMOPROG\C*.BAT	*C* batch files

Installing dGE screen drivers

Latest release information is contained in the file README.1ST. Please view this for any change in instructions before installing *dGE*.

1. Create a directory on your target hard or floppy disk and change directories (CD) into it. We recommend you call it \DGEV2 which we will refer to later in the examples on running the demo programs.

2. Place *dGE* Disk 1 in drive A and log onto this drive.

3. Run the batch file DINSTALL.BAT.

   A:>DINSTALL *Drive DriverType* ↵

where	*Drive*	target drive e.g. C	
	DriverType	EGA	for EGA
		CGM	for CGA mono
		CGC	for CGA colour
		HRC	for HERCULES mono

For the IBM PS/2 model 30 use the CGA drivers. For the IBM PS/2 models 50, 60 and 80 use the EGA drivers.

Installing the dGE host program interface

Following the installation of the driver, replace Disk 1 with Disk 2 and run the batch file HINSTALL.BAT. Note that if you received *dGE* on a 3.5 inch disk you do not have to replace it - all the files are on one disk.

A:>HINSTALL *Drive HostProgram* ↵

where	*Drive*	target drive e.g. C	
	HostProgram	DB3	for *dBASE III*
		CLIP	for *Clipper*
		FOX	for *FoxBASE+*
		C	for *C*

The installed system

The installed system is created in the directory nominated during the installation procedure. There are no subdirectories below the original directory.

README.1ST	latest information on *dGE*
BUGS.DOC	latest bug report
SETDGE.COM	graphics kernel for your screen
*.CHR	character files
VIEW.COM	view utility
CONFIGP.EXE	matrix printer configurator
HGC.COM	HERCULES mode switch

dBASE III

DGE.PRG	linkage for *dBASE III*
CONFIG.DB	*dBASE* config file

Clipper

DGEFUNS.PRG	linkage for *Clipper*
DGECLASM.OBJ	second link module
CLIPCOMP.BAT	compile and link batch file

FoxBASE+

DGEUDFS.PRG	linkage for *FoxBASE+*
DGE.BIN	second part

C

DGEC.C	linkage for *C*
DGECASM.ASM	second part
DGEC.H	header definition file
MSCCOMP.BAT	compile and link batch file

dBASE language

DGEDEFS.PRG	*dGE* public definitions for *dBASE*
*.PRG	example programs
*.DBF	example database files
*.BAT	example batch files

C language

*.C	example programs and data
*.H	
*.DAT	
*.BAT	

Running the demonstration programs

dBASE III

1. Log into the directory \DGEV2.
2. Ensure that *dBASE* is available via a path.
3. Run the batch file DB3DEMO.BAT.

This will execute the suite of *dBASE* programs starting with the module DB3MENU.PRG.

Clipper

1. Log into the directory \DGEV2.
2. Ensure that the *Clipper* compiler and PLINK86 are available via a path.
3. Run the batch file CLIPDEMO.BAT.

This will compile and link the demo programs and linkage module. After this it will execute the compiled demonstration program CLIPMENU.EXE.

FoxBASE+

1. Log into the directory \DGEV2.
2. Ensure that *FoxBASE+* is available via a path and that is is invoked by the name FOXPLUS.
3. Run the batch file FOXDEMO.BAT.

This will execute the suite of *FoxBASE+* programs starting with FOXMENU.PRG.

C

A batch file exists to enable you to compile and link a *dGE* application using Microsoft C. If you are using a different *C* compiler edit the batch file to invoke the compilation appropriately.

1. Log into the directory \DGEV2.

2. Ensure that MASM.EXE and your *C* compiler are available via a path.

3. Run the batch file CDEMO.BAT.

This will compile and link the demo programs and linkage module. After this it will execute the compiled demonstration program, CMENU.EXE.

Graphics co-ordinates

The layout and operation of the graphics screen differs from the screen in text mode. In text mode, the screen is composed of cells organised in lines and columns. Each cell is capable of displaying a single character. Typically a text screen contains 25 lines of 80 characters.

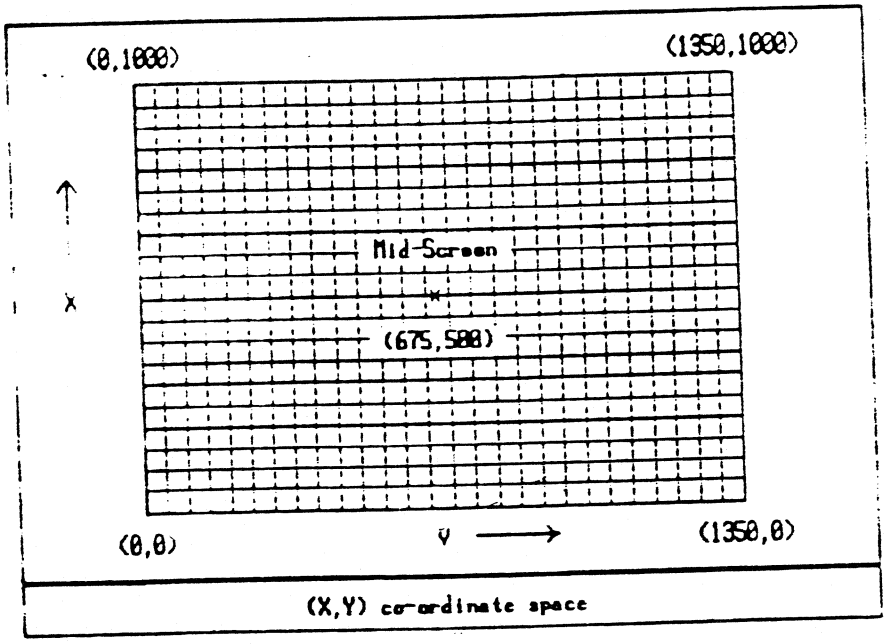
The graphics screen in contrast is made up of a large number of picture elements or pixels, again organised in rows and columns. The EGA screen has 350 lines of 640 pixels. Each pixel can be illuminated or extinguished, so that complex images can be created.

In order to standardise *dGE* between diferent video standards the screen in *dGE* has been defined as 1000 units high. That is to say there are 1000 lines which can be addressed in the Y direction, (vertically). A unit on the screen is called a *screen unit*. When referring to co-ordinates within *dGE* we will always be using these units.

The IBM screen has an approximate aspect ratio of 1:1.35. That is to say that the screen width is 1.35 times its height. As a result, the width of the screen is 1350 screen units. The origin, X=0 Y=0, is in the bottom left corner, and the maximum co-ordinate, X=1350 Y=1000, at the top right.

The *unit length* on the screen in both the X and Y directions is the same, that is, a box with width 100 units and height 100 units will be represented as a true square with sides of equal length. Similarly, circles will appear as true circles and not ovals. If circles drawn by *dGE* appear oval adjust the vertical height control on your monitor.

dGE maps its co-ordinate space down to the resolution supported by your graphics adapter. Currently there is no IBM video standard that exceeds this co-ordinate space.



Both *dGE* graphics and text commands use this co-ordinate arrangement.

Graphics Adapter	Screen Size (Pixels) Horizontal	Vertical	Dot File
HERCULES	720	348	32K
CGA mono	640	200	16K
CGA colour	320	200	16K
EGA colour	640	350	112K

Table 3.1 Graphics Adapter Details

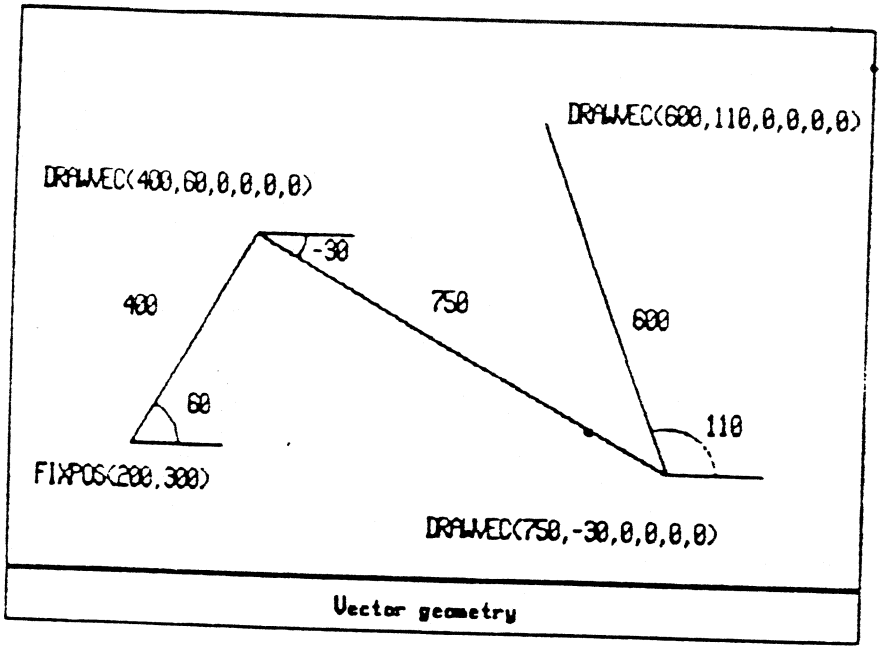
The Dot File is created when you save the screen image in a disk file. Note that the EGA version uses data compression to save space. On average an EGA dot file will be about a quarter of the size indicated.

Vectors

dGE introduces a number of functions that use vector variables. A vector variable is defined in terms of length and angle.

Vector geometry presents an alternative to the more familiar cartesian or X-Y geometry. In all vector functions angles are in degrees (-360 to +360) with 0 in the positive X direction and the angle increasing in the counter-clockwise direction. The unit of length is the same as in the cartesian form.

As an example a vertical vector extending from the bottom left corner of the screen to the top left will have length 1000 and angle 90 degrees.



In vector geometry an emphasis is placed on *relative* moves and draws.

Vector lines are drawn from the *current position* to a final position determined by the magnitude and angle of the vector. On first entering a program the current position is at the bottom left of the screen, that is 0,0.

Thereafter as vector functions are executed the current position is moved to the final point of each vector. The current-position may be re-defined at any time at an absolute X,Y position using the **fixpos**(..) function.

Character sets

Characters are drawn in cells similar to characters drawn in text mode. However, unlike text mode, characters can be drawn anywhere on the screen.

dGE allows text to be displayed using different character sets or fonts. At any one time *dGE* can directly access two fonts held in memory, since it has two buffers to hold them. The required memory is reserved by the program SETDGE. These fonts are quite separate from the characters used in text mode. *dGE* has no built-in character set.

Before any *dGE* text commands are issued a character set must first be loaded into memory. Failure to do so will result in ? appearing on the screen in place of characters.

Character sets are contained in files with the extension . CHR. Use the **loadcset**(..) function to load a character set into one of the two character set buffers.

All *dGE* text commands require you to specify the character set buffer (0 or 1) as a function parameter.

Graphics Adapter	Cell Size (pixels) Horizontal	Vertical	Characters Per Line
HERCULES	10	16	72
CGA mono	9	8	71
CGA colour	8	8	40
EGA colour	9	16	71

Table 3.2 Text cell sizes

When an image is drawn on the screen it will be drawn over the background in one of three foreground colours, selected as colour 1, 2 or 3.

The CGA board supports two foreground palettes. Hence, at any one time, the foreground colours can be:

1	GREEN
2	RED
3	YELLOW

when palette 0 is selected. Or:

1	CYAN
2	MAGENTA
3	WHITE

when palette 1 is selected.

dGE has a simple function to select the current background palette, the foreground colour and the foreground intensity -

`setpal` (*Background, Intensity, Foreground*)

Note that the intensity attribute may have no effect on some screens.

Character sets

The colour character sets are the same as for the mono version of *dGE*. However, owing to the reduced resolution of the IBM colour screen the characters appear wider. Only 40 characters per line can be written as compared to 71 in the mono version. This is the only significant programming variation between the two versions.

Note that the `getchar()` function returns a driver identifier which can be used to switch between driver dependent code.

EGA Adapter

The EGA adapter, fitted with minimum of 128K of memory, provides for 16 foreground colours and up to 64 background colours. The values of the colour attributes are the same as those for the CGA adapter.

The foreground colours are selected by the colour attribute in the function call.

The background is selected from the full EGA palette using the **setpal(..)** function. It remains in force until the next **settext()** when it is reset to black.

The colour parameter when calling dGE functions

Text and images drawn by *dGE* are in one of the current foreground colours, as selected by the *dGE* function.

```
Colour = 3
DRAWLINE (x0, y0, x1, y1, 0, 0, Colour)
```

will draw a line in colour '3'.

Note that if colour 0 is selected in the *dGE* function the colour is unaltered, that is, it assumes the same colour as the last function drawn.

Use of the colour attribute parameters is documented for each function in the manual and quick reference sections.

On mono screens the colour parameter is ignored. However, mono screen operation does not restrict you from plotting in colour.

The relationship between the *dGE* colours selected for a function and those produced on a plotter is entirely within your control. The function **plotpen(..)** defines the relationship between the *dGE* colour and the pen number.

Chapter 4

Linking dGE with the host program

dGE Organisation

dGE consists of two distinct parts -

- The memory-resident graphics kernel, SETDGE, which is common to all host programs.
- The host-program interface.

This chapter discusses the use of both of these components.

The Graphics Kernel - SETDGE

There are versions of SETDGE for each of the screen types, EGA, CGA mono, CGA colour and HERCULES. Functionally they operate in the same way. SETDGE must be loaded in memory prior to running your applications program. To do this type:

SETDGE↵

A message informs you that it has loaded successfully. Now you may run your *dGE* applications program.

After the program has finished you may free the memory used by SETDGE by typing:

SETDGE /F↵

Note, however, that memory may not be freed if you run another memory-resident program after SETDGE. In this situation DOS is unable to re-allocate the memory.

A switch may be used to start up *dGE* in the Version 1 compatibility mode. Type:

SETDGE /1↵

This will enable you to run *dGE Version 1* programs unaltered. Compatibility mode can also be selected within the applications program using the **setver(..)** function.

The interface modules

When you run your host *dBASE* (or other) application, you will need to include one or more interface modules to enable *dGE* functions to link to the graphics kernel.

Here we describe the interface modules for the various host programs currently supported.

dBASE III

dGE functions within *dBASE III* are macros defined in a file DGE.PRG. These macros expand into print statements which directly output ESCAPE sequences to the resident module, SETDGE.

At the start of your main program (e.g. MAIN.PRG) include the statement:

```
DO DGE
```

This will initialise the *dGE* functions.

The *dGE* macros are PUBLIC memory variables. If you wish to reduce the number of memory variables used by *dGE* you may edit DGE.PRG and remove unwanted functions.

Clipper

dGE functions within *Clipper* are UDF's defined in a file DGEFUNS.PRG. These functions communicate with SETDGE via an object file DGECLASM.OBJ.

Formerly both these modules were combined in a library file to link with compiled *Clipper* programs. This practice has been discontinued because of the frequent changes in object formats. Instead you are provided with the source file of the function module to compile with your own version of the compiler. Compile this as follows -

```
CLIPPER DGEFUNS↵
```

which will produce the object file DGEFUNS.OBJ.

The second module DGECLASM.OBJ must also be linked with your main program. After compiling your program (e.g. MAIN.PRG) as normal link it with *dGE* as follows:

```
PLINK86 FI MAIN FI DGEFUNS FI DGECLASM LI CLIPPER↵
```

Note that DGEFUNS.PRG can be edited to remove unused *dGE* functions, thereby reducing the size of the object file. Also note that DGEFUNS.OBJ and DGECLASM.OBJ may be combined in a library if this is more convenient for you.

FoxBASE+

dGE functions within *FoxBASE+* are UDF's defined as procedures in a procedure file DGEUDFS.PRG. These functions communicate with SETDGE via a binary file DGE.BIN.

To create the linkage place these two statements at the start of your main program -

```
LOAD DGE
SET PROCEDURE TO DGEUDFS
```

The procedure file may be compiled by FOXPCOMP in the same way as your applications program to produce faster execution.

Note that the two functions, **drawcircle(..)** and **polargraph(..)** have been abbreviated to **drawcirc(..)** and **polargr(..)** because of an evident inability of *FoxBASE+* to cope with function names of 10 characters.

C language

The interface provided by the files DGEC.C, DGEC.H and DGECASM.ASM enables you to call *dGE* functions from within your *C* program. Take the following steps to link *dGE* into a *C* program.

1. Write the host program with DGEC.H included at the start of each module. This provides forward declaration of the *dGE* functions and also defines some symbolic names to aid programming of *dGE* parameters.

2. Generate the two object files DGEC.OBJ and DGECASM.OBJ with the following two commands –

   ```
   MASM DGECASM;↵      (Microsoft Macro Assembler example)
   MSC DGEC;↵          (Microsoft C example)
   ```

   The assembler module, DGECASM, must be configured to match the memory model of your host program. This is done by editing the equated value of `@largemodel` at the start of the file DGECASM.ASM.

   The compilation of DGEC.C will require DGEC.H and STDIO.H to be available. The only functions used from the latter are **sprintf** and **scanf**, which must be in the *C* support library.

3. Compile the modules of your program as appropriate.

4. Link your programs object file to the *dGE* files, e.g.

   ```
   LINK yourprog+dgec+dgecasm;↵
   ```

5. To run your program, first load the graphics kernel by running SETDGE.

   ```
   SETDGE↵
   ```

   Then run your program as normal.

If you have any problems, look at and, if necessary, edit the source files DGEC.C, DGECASM.ASM, DGEC.H and the demonstration program sources – having first taken backup copies !

In *C* and *Clipper* it is not necessary to assign the function to a variable. ie. the statement:

```
functionname(...)
```

will suffice if no useful result is returned by the function.

dBASE III

Functions are written in the form:

```
&functionname, Param1, Param2, ... , ParamN, end
```

where -

functionname	is the symbollic name of the *dGE* function.
Param1,2 etc.	are numeric or string parameters.
end	is a delimiter.

String parameters must be preceded by `,txt,`.

```
&functionname, Param1, txt, Param2, ... , ParamN, end
```

where Param2 is a string.

`end` and `txt` are public memory variables defined in `DGE.PRG`.

In the few cases where functions return meaningful values you must follow the function declaration with an accept statement.

```
&functionname, Param1, ... , end
accept to MemVar
```

will read a returned variable into `MemVar`.

dGE uses the keyboard buffer to read variables. It is recommended that you flush this buffer before calling this function. Also, it is preferable to set console off while accepting the returned value.

```
do while inkey<>0
enddo
&functionname, Param1, ... , end
set console off
accept to MemVar
set console on
```

Constructing a program module

dGE graphics can be introduced at any point within a program. Graphics commands might be localised within a region of a single program module or, as is more likely, be separated into discrete, graphics-oriented program files.

In *Clipper*, *C* and *FoxBASE+* a complete graph drawing module may be written within a function, with parameters passed to it in the function command line.

To enter graphics mode the command **sethires**(n) must first be issued, where n is a parameter determining screen number.

```
R = sethires(0)
```

From this point on the screen is switched via the operating system into graphics mode. Normal text may no longer be displayed, and attempting to do so will cause unpredictable results.

status, **talk** and **scoreboard** all produce text output as well as explicit **?**, **@ say** or **@ get** commands.

Next you will probably want to clear the screen.

```
R = clrscreen()
```

The *dBASE* **clear** command is also text-based and must not be used.

Any mixture of *dGE* and *dBASE* non-text commands may now be used until you wish to return to text mode.

This is achieved by:

```
R = settext()
```

A *dGE* program module will typically look like this:

```
set talk off
set scoreboard off
set status off
set console off          && dont do this in dBASE

r = sethires(0)
r = clrscreen()
  ..
  ..
  dGE and dBASE commands
  ..
  ..
r = settext()
set console on           && dont do this in dBASE
clear
return
```

By far the easiest way to learn how to write the bit in the middle is by example, and you are referred to the abundant demonstration programs provided.

A program, SKELETON.PRG, is also provided to assist you.

Parameter passing

Passing numeric parameters

Most *dGE* functions involve the passing of parameters, for example screen co-ordinates, data values etc.

Consider the simple example of selecting the screen for graphics mode. The HERCULES adapter allows more than one screen to be used.

The command is:

sethires(0)	Selects SCREEN 0
sethires(1)	Selects SCREEN 1

The parameter list for each function is fully described later.

Note the following rules relating to function parameters:-

1. Numeric parameters may be positive or negative numbers in the range -32,768 to +32,767.

2. They may be

   a) Literal numbers e.g. `99`
   b) Memory variables e.g. `Xorigin`
   c) Fields from a database e.g. `UnitPrice`
   d) Expressions containing any of the above e.g. `99+Xorigin` OR `1.15*UnitPrice`

3. Fractional parts of numbers are discarded. e.g. 99.9 is rounded down to 99. Add 0.5 to variables/numbers to round to nearest integer, (e.g. 99.9+0.5 = 100.4 rounds to 100) or use the round(value,0) function.

   In functions that involve *relative* movement and drawing, numeric parameters are rounded within the function (e.g. in DGEFUNS.PRG) to minimise cumulative errors. Note that you are at liberty to introduce rounding into other functions by editing the interface library. This does not apply to *dBASE*, where rounding must be done in your application program.

With these rules in mind consider the example of drawing a line on-screen from the co-ordinate X=10,Y=10 to X=100,Y=100 and to use the default of drawing a white unbroken line.

The definition of this function is:-

`drawline(` *X1, Y1, X2, Y2, LineMode, LineStyle, Colour* `)`

Where:

X1,Y1	are the co-ordinates of the initial point
X2,Y2	are the co-ordinates of the final point
LineMode	0 for an overwrite line
LineStyle	0 for a solid (not dashed) line
Colour	7 for white on the EGA

The function line may be written as

```
drawline(10,10,100,100,0,0,7)
```

Passing string parameters

So far we have seen how numbers are passed to *dGE*. There are also several text commands which use character strings. Strings are passed to *dGE* in the parameter list. They may be literals (e.g. 'NAME') or fields or memory variables that have been assigned to strings.

The function to write a string on the screen is saystring(..).

The statement to write a string at X=10, Y=10 is:

```
String = "This is a message .."
saystring(10,10,0,0,0,String)
```

Or

```
saystring(10,10,0,0,0,"This is a message  ..")
```

The co-ordinates can be described by memory variables.

```
X = 10
Y = 10
saystring( X,Y,0,0,0,"This is a message ..")
```

dGE uses characters defined in a .CHR character file. Such a file must first be loaded using the **loadcset**(..) function. Two character sets may be loaded and active at any time.

String parameters returned from dGE

The complement to printing a string on the screen is reading a string from the keyboard in response to a query. In *Clipper* we have the command @..get. In *dGE* the command is **getstring**.(..).

In this case we expect to receive a string in return. The procedure is similar to **saystring**(..). The position of the string on the screen is delimited by colons.

The command line is -

```
Response = getstring( 100, 200, 1, 10, 5 )
```

Note that in this case the function does return a useful parameter, namely the string, **Response**, read from the keyboard.

There a several other *dGE* commands that return string variables to the caller.

Note that *dBASE III* requires the **accept to** .. statement.

C language interface

Function parameters

The values of function parameters are described in the main part of this documentation. All parameters have defined types – *dGE* will not work properly if variables of the wrong

type are put in the parameter list of a function call, e.g. real instead of integer. If in doubt, ensure all parameters are cast to the correct type. For example –

```
datastore( (int)(base+data1*14), 0, 0, 0);
```

All function parameters are of integer type, `(int)`, except for the following –

1. File names for disk operations.

   Here the variable *fname* is always a character pointer, `(char *)`.

   `diskfile(` *mode, fname* `);`
   `loadcset(` *cset, fname* `);`
   `printfile(` *mode, fname* `);`

2. String and label printing and editing strings.

   For these functions the variable *string* is always a character pointer, `(char *)`.

   `saystring(` *x1, y1, cset, mode, col, string* `);`
   `vecstring(` *len, ang, cset, mode, col, string* `);`
   `edstring(` *x1, y1, cset, col, string* `);`
   `labelx(` *x, y, inc, len, cset, mode. col, string* `);`
   `labely(` *x, y, inc, len, cset, mode, col, string* `);`
   `labelpie(` *xlen, rad, len, cset, mode, col, string* `);`

3. Icon handling functions.

   These operate in one two modes – either with a selection from the internal icon library, or with a user defined icon. In the latter case the parameter *icon* is a string pointer to a description of the user icon.

   In the former case *icon* is an integer selecting the internal library icon. However, to preserve typing in the *C* parameters this integer must be in the form of a string, the pointer of which is passed in the variable *icon*.

   `vecicon(` *len, ang, mode, icon, col* `);`
   `sayicon(` *x1, y1, mode, icon, col* `);`

Function return types.

dGE functions generally return integer values. Even those functions without a sensible value to return, return zero. For some functions, however, returning an integer value is inappropriate. These exceptions are listed below.

1. The functions which return statistics derived from the data in *dGE*'s buffer clearly need a higher level of accuracy than can be achieved with integers. These functions return floating point values. These functions are –

```
getsin( ang,mode);
getasin( val,mode);
getcc();
getmax();
getmean();
getmin();
getsd();
```

2. The keyboard functions **getchr**, **getstring** and **edstring** need to return characters or character arrays, i.e. strings.

   a) **getchr** simply returns a character of type `char`.

   b) **getstring** returns a pointer to an internal buffer which can hold up to 80 characters. This buffer is used on the next call to **getstring**, so if the string is to be retained it must be saved by the host program. The following code will give problems.

```
char *a, *b;
a = getstring( x, y, cset, nchars, colour);
b = getstring( x, y, cset, nchars, colour);
```

   The string pointed to by **a** will be altered by the second call to **getstring**. The correct form is as follows –

```
char a[80], b[80];
strcpy(a, getstring( x, y, cset, nchars,
                     colour));
strcpy(b, getstring( x, y, cset, nchars,
                     colour));
```

To avoid these problems you are advised to use **edstring** instead of **getstring**.

c) **edstring** takes a pointer to an array of characters provided by the host program and edits that array directly. It also returns a pointer to that same array. The correct form for using **edstring** is as follows –

```
char a[10];
strcpy( a, "default ");
edstring(x, y, cset, colour, a);
```

The edited string is held in **a**.

Please note that a call in the style –

```
result = edstring(x, y, cset, colour,
                  "Edit this");
```

will cause **edstring** to directly alter the string literal. This may corrupt the *C* language environment and therefore the practice is not recommended.

Chapter 6

Data presentation in graphs and charts

Overview

Graphs, charts and polylines are drawn by functions which use data previously stored in a *dGE* internal array. The **datareset()** function clears the array in preparation for loading new data. Each ensuing **datastore(p1, p2, p3, p4)** statement loads data and flags for a "datapoint".

The parameters have different meanings depending on the drawing function. For example in an X-Y graph **p1** is the amplitude in screen units, **p2** is the code identifying the symbol for the point and **p3** is optionally the X position in screen units.

When all the points are loaded, usually using a **do .. while** loop, the drawing function may be executed.

```
xygraph( 100, 100, 50, 1, 2)
```

where 100,100 is the origin, 50 is the increment in X, 1 and 2 specify the style and colour of the graph. By default all the data in the internal array is displayed.

A basic graphing operation

1. issue a **datareset()** function call. This clears the *dGE* data buffer of any data accumulated for a previous graph.

2. For each data point issue a **datastore(..)** function call, placing the value and attributes to be accumulated in the parameter list.

3. When all the data is transmitted issue a function call to draw the graph or chart.

```
datareset()
datastore(10,1,0,0)
datastore(20,2,0,0)
datastore(30,3,0,0)
piechart(675,500,250)
```

This will draw a pie chart with centre at (675,500) and radius 250, having 3 segments in the ratio 10:20:30. The first segment is filled with pattern 1, the second with pattern 2 and the third with pattern 3.

This is a simple example, but it illustrates practically all you need to understand about the method of depicting data in graphical form.

It will be more efficient to transmit a large number of data values using a **do .. while** loop. For example, when loading data from a file –

```
do while.not.eof()
  datastore( p1 , p2, p3, p4 )
  skip
enddo
```

Also you will need to scale your data, draw axes and labels, and apply method to the layout of the screen.

Ranging data

By default a graphing function shows all the data in the *dGE* internal array. ie. all data accumulated since the last **datareset()** function call.

The **datarange(..)** function allows you to nominate a range of points to show. That is to say, a subset of the complete array can be selected.

`datarange ( `*firstpoint, lastpoint* `)`

firstpoint	identifies the first point to show.
lastpoint	identifies the last.

The array is indexed from zero. The first point is index 0 and the last is (n-1) where n is the number of points in the array. This is consistent with the graphing functions, which show the first point at X = 0.

Data ranging also applies to the statistics functions. By nominating ranges of data it is possible to "move" through a long graph superimposing moving means and best-fit lines. The range of data is always reset to the full array following a **datareset()**.

Re-scaling data

The *dGE* data array stays active until a **datareset()** is issued. Hence it is possible to use the same data set to draw a variety of graphs without re-loading the data.

The **datapc(scalefactor)** function allows you to instantly re-scale the data in the array prior to drawing it. It has particular application in the drawing and re-drawing of polyline diagrams.

scalefactor is a percentage re-scale factor to be applied to the data.

```
datapc(100)
```

would set the scale factor to 100% (unity) which is the default.

```
datapc(50)
```

would reduce data to 50%.

The % scale factor can be in the range 1 to 10,000 The scale factor is always restored to 100% following a **datareset()** function call.

The Resume feature

Users of *dGE Version 1* have remarked on the limitation that all the data had to be **datastored** before the graph could be drawn. In practice this could result in long periods of inactivity while large datasets were loaded in **do .. while** loops.

In Version 2 the *resume* feature has been introduced. This allows the graph to *resume* from where it last finished. It is now possible to call the graphing function after each data point is added.

```
datastore(..)
datastore(..)
datastore(..)
xygraph(..)
```

dGE Version 1

```
datastore(..)
xygraph(..)
datastore(..)
xygraph(..)
datastore(..)
xygraph(..)
```

dGE Version 2 resume

Clearly the whole process of drawing the graph will take longer using the latter technique

since more function calls are made. However, it is for you to decide whether this time penalty is worth the benefit of seeing the graph build up on screen.

Note that the resume technique is not applicable to those graphing functions where the entire data set is required for analysis prior to drawing, e.g. Pie charts and fixed increment polar graphs.

The resume feature is always enabled and requires no special action on the part of the user

Time series graphs

dGE Version 2 introduces an entirely new kind of graph. Since it is used most frequently for data plotted against time it is referred to here as a "time series", or "time history" graph. It operates in the opposite manner to all other *dGE* graph functions. The graphing function is called first, after which points are added by calling the equivalent of the **datastore**(..) function.

```
timegraph(..)          && initiate the graph
timedata(..)           && add the first point
timedata)..)           && add the second point
  ..etc.               && continue indefinitely
```

The **timegraph**(..) function defines the position of the graph on the screen and the nature of the data to be portrayed. Subsequent **timedata(..)** calls add data points.

The first point is drawn at the origin of the graph on the right of the screen. As data points are added the graph scrolls to the left and the new point is added at the origin. The result is an animated graph that moves continually to the left. Time "increases" to the right

The *dGE* data buffer is circular. As new points are added the oldest is dropped. This process can be continued indefinitely.

Up to four time coincident *channels* can be portrayed. One or more of these channels can be nominated to be an *index* channel. An index channel is both a datum line and a means of applying regular timescale marks to the moving data.

Moving mean and standard deviation lines can be superimposed on any or all of the channels.

Summary of dGE syntax and operation

- Function statements always start with a function name followed by zero or more parameters. The function may or may not return a useful result.
- Each function requires a fixed number of parameters. It is important that the exact number of parameters is used otherwise a syntax error occurs. Also ensure that parameters are of the correct type .. date variables are not the same as character variables.
- Numeric parameters must be integers in the range -32,768 to +32,767. They may be literal numbers, numeric memory variables, numeric fields from a data file, or expressions containing any combination of these.
- String parameters may be literals, memory variables, fields from a data file, or expressions containing any combination of these.
- A character set must be loaded using the **loadcset(..)** function before text commands are issued from *dGE*.
- To present data on screen using the graph and chart functions it is first necessary to transmit the data set using the **datareset()** and **datastore(..)** commands. When the data set has been transmitted you issue the command to draw the chart.
- Data can be ranged and scaled at any time. Ranging and scaling apply to the current data set until a **datareset()** call is made. Ranging and scaling also applies to the statistics and best-fit functions.
- Graphs can be *resumed* at any time during a long series of datastore's
- Time series graphs begin with the **timegraph(..)** function followed by an unlimited number of **timedata(..)** calls. Scaling applies but ranging does not.

Chapter 7

Hardcopy

Re-direction to file

dGE Version 2 introduces the ability to direct hardcopy output, (whether to printer, laser or plotter), into a named file. The command **printfile(..)** is used to enable and disable re-direction. Once re-direction is enabled all hardcopy output is directed to that file whenever one of the hardcopy functions is invoked.

With output to printers the option exists to overwrite or append to the file. Hence a single file may be used to collect the output of several screens. With plotters, output is always appended since it is inherently a discontinuous activity, (it is an indefinite sequence of plotter commands).

Be careful not to mix types of output. Plotter output is not likely to produce sensible results on a matrix printer.

atrix printers

dGE has a driver for printing the pixel image of the screen on a matrix printer. This is invoked by the **printscrn()** function. The matrix printer must be capable of responding to EPSON graphics codes or codes closely related.

The utility program, CONFIGP.EXE is used to alter the parameters in your copy of SETDGE.COM. First ensure that both of these programs are in the same directory on disk. Then type:-

CONFIGP↵

A list of printers will appear for you to select from. If your printer is in the list then select the number and proceed to answer some simple questions before saving the changes. If not you will have to define your own codes, using the technical information in your printer manual.

Printer control parameters

To view current parameters in your copy of SETDGE.COM select 100 and press ↵.

```
Current printer parameters :-

                                Horizontal
                                Bit 0 at bottom
Page orientation
Pin polarity                    27   65   8
Pre graphics-page control codes 27   42   6   128   2
Pre graphics-line control codes 13   10
Post graphics-line control codes 13  27   50  12
Post page control codes
```

These are the default parameters for the EPSON FX on an IBM PC using a CGA card.

Page Orientation

You have the option to print the picture horizontally across the page or vertically.

Pin Polarity

Matrix printers use a print head consisting of a vertical row of pins. The *dGE* printer driver demands that there are eight pins. These are numbered 0 to 7. Your printer manual will advise you whether Pin 0 is at the top or the bottom of the head.

Note that printers with nine or more pin heads can be used as long as they have a graphics mode that has an eight pin mode. Most modern 24 pin printers can operate in such a mode.

Pre-graphics page control codes

The control codes shown in the preceding table are the decimal values of ASCII characters which must be transmitted to your printer to alter its mode of operation. The pre-graphics page control codes are codes which alter the behaviour of your printer for the duration of printing the picture on your screen. Typically the most important mode to set is the line spacing.

In character mode the line spacing is usually $^1/_6$ inch. However, in graphics mode the line spacing must be set to the physical height of the matrix head. This is required so that successive rows of 8 vertical dots follow each other down the page without overlapping or leaving a gap.

In the example above the codes are -

```
27, 65, 8
```

This is equivalent in ASCII to ESC, "A", 8. Note that ESC is a special ASCII control character while 8 is the numeric *value* of the last byte in the sequence. This sequence selects $^8/_{72}$ inch line spacing on an EPSON printer. If your manual does not give the physical height of the matrix head it may be necessary to experiment with different spacing.

Note that since the picture is a fixed number of rows high (the CGA has 50 rows) the physical height of the picture is fixed on the page. The width, and hence the aspect ratio may be varied by the pre-graphics line control codes, which set the horizontal density of printing.

On some printers the horizontal density is also set by the *characters per inch* codes (e.g. 10

cpi, 12 cpi). If so, the appropriate control codes should be included here following the *lines per inch*.

Pre-graphics line control codes

The principal purpose of these codes is to set the printer into graphics mode to receive a row of graphic *bit-image* bytes. These are of the form -

<graphics-control-codes> <number-of-bytes-in-a-row>

In the example above the control codes are -

```
27, 42, 6
```

= ESC, "*", 6 which for an EPSON sets graphics mode with a horizontal printing density of 90 dots per inch.

The number of bytes per row are -

```
128, 2
```

This is the EPSON code to indicate 128 + (2 x 256) = 640 bytes per row.

The complete set is thus -

```
27, 42, 6, 128, 2
```

CONFIGP informs you of the number of bytes per row when you reach the point of entering this information.

Note that the aspect ratio of the printed picture can only be set by varying the horizontal dot density. By reducing the dot density the picture is widened, by increasing it the width is reduced. Obtaining a good aspect ratio is not crucial, but circles will appear elliptical unless a reasonable density can be found. Printers rarely have the ideal dot density. Consult your manual for the range available and experiment with them.

Newer machines can often be run in EPSON mode. Check if this is so, and if a more suitable density is available in this mode.

Some older models of printers are incapable of accepting graphics information in the form *<graphics-control-codes> <number-of-bytes-in-a-row>*.

SUCH PRINTERS CANNOT UNDER ANY CIRCUMSTANCES BE CONFIGURED FOR dGE.

An additional use of pre-graphics line control codes is to insert spaces or tabs at the beginning of each line to centre the picture on the page. These may be entered before the graphics control codes.

Post-graphics line control codes

All that are required are CR and LF.

```
13, 10
```

Post-page control codes

Usually CR and FF and control codes to reset line spacing for text (e.g. $^1/_6$ inch).

```
13, 27, 50, 10
```

= CR, ESC, "2", FF

Troubleshooting

1. If your printer churns out row upon row of random characters it is almost certain that the pre-graphics control codes are incorrect for the machine .. the graphics control codes are not switching the machine into a graphics mode. It is known that some printers are supplied with variations in their ROM software, altering or removing documented graphics modes - the OKI Microline is an example. Double check that your codes are valid for the printer. If all else fails investigate whether you have such a variant.

2. If you get a mixture of random characters and graphics you have probably specified the wrong number of bytes in a row.

3. Spaces between adjacent rows, or compression (overprinting) of rows indicates a wrong line spacing.

Printer Command Language devices

dGE has a driver capable of printing a raster image of the screen on a printer that supports the PCL (Printer Command Language) developed by HEWLETT-PACKARD and implemented on the Laserjet and other printers.

The **printpcl(..)** command directs an exact dump of the screen raster image to the attached printer or print file. Command parameters determine the horizontal and vertical margin, the size of the image and the addition of special codes such as Reset and Form Feed.

Laser printers have a consistent dot density horizontally and vertically on the page. IBM screens, however, do not. As a result the image as printed will be slightly elongated, leading to elliptical circles.

An option may be selected to *stretch* the image to produce the exact aspect ratio of a screen image on the page. The disadvantage of this is that undesirable interference effects can be introduced, particularly in regular patterns, since every so often a line must be doubled up.

Plotters

Graphics output may be directed to a plotter or plotter file. The plotter driver adheres to the HEWLETT-PACKARD HP-GL standard.

Usually you will want to preview the plotter output on the graphics screen, and graphics output will occur concurrently to both devices if **sethires**(..) and **ploton**(..) are called. Note however that plotter output may be obtained without a graphics screen.

If you wish to create plotter graphics on a mono screen use the HERCULES driver and omit the **sethires**(..) and **settext**() commands. After the **ploton**(..) function is called all appropriate drawing functions are output to the plotter. This continues until the **plotoff**() function is called.

Not all *dGE* drawing functions are appropriate for plotting: clearing and dragging of polylines is illogical on a device that is incapable of erasing images, and functions such as these should be avoided.

Certain raster-based features of *dGE* such as its character sets and fill patterns have to be handled in a different manner to the screen. *dGE* uses the internal fonts of the plotter which are selected by the **plotcset**(..) function. Fill patterns are also taken from those internal to the device.

The relationship between colour and the pen used is set by a function **plotpen**(..).

Chapter 8

The operating system environment

Memory overhead

SETDGE captures about 35 Kbytes of memory when it is loaded. Other than this, the only memory overhead is the additional size of the .EXE file in compiled programs due to *dGE* code and interface modules.

dGE interface modules can be reduced in size by deleting any references to functions you are not using.

After you have finished running a *dGE* program you can recover the memory used by SETDGE by re-running the program at with the FREE switch.

Type -

```
SETDGE /F↵
```

This will remove SETDGE from memory and release the memory occupied by it. However, if you run another terminate-and-stay-resident program after the inital installation of SETDGE, then DOS will be unable to reuse the freed memory.

Networks

dGE operates in a network within the abilities and limitations of the host program. *dGE* accesses only a limited range of files directly. These are -

- Character set files
- Screen Image files
- Print files

dGE is not capable of applying network locks to these files. If locks are required it may be possible to apply them indirectly using features of the host program.

Microsoft Windows

dGE will work within Microsoft Windows. The simplest procedure is to run SETDGE.COM before running Windows so that SETDGE does not require a .PIF file. Alternatively it can be loaded through windows as a graphics TSR process.

The host program (*dBASE*, *FoxBASE+* or .EXE file) will require a modified .PIF file to enable it to generate graphics.

When editing the .PIF file for your application you must specify -

- Graphics/multiple text under the Program Switch option, and
- Graphics/Text under Screen Exchange.

This will allow your application to create graphics and save the screen when switching contexts.

OS/2

dGE will work in OS/2 in the DOS Compatibility box. This mode of operation emulates the *real* mode of DOS.

Chapter 9

Programming hints

Publicly defined variables

Many *dGE* functions have arbitrary numeric parameters defining mode of operation, style and colour. To simplify programming, a library of publicly defined variables is provided which can be used in parameter lists. The files containing these public variables are:

DGEDEFS.PRG	for the *dBASE* language
DGEC.H	for the *C* language

In the *dBASE* language environment include the statement, DO DGEDEFS, at the beginning of your program. This initialises the public variables.

For the *C* language environment, use the `#include` directive to include the file DGEC.H at the start of every program module.

You are at liberty to modify these files to suit your needs. They are not a formal part of *dGE*.

The example is a complete program module excluding the loading of the *dGE* linkage programs (if necessary) and the setting of text functions off. It is written in *Clipper/FoxBASE+* style. Users of *dBASE* should note the difference in syntax, principal & instead of brackets, the ,txt string delimiter and ,end closing delimiter.

The lines 1 to 4 set the screen into graphics mode, load a character set and print a title o the top of the screen. The +8 mode in the **saystring(..)** function centres the title aroun X=675 at the top of the page. Note that, by specifying X=675, the string is actually centred on the screen.

The next four statments, beginning with line 5, define the size and location of the graph the screen. The origin at the lower left is at (150,200), the height is 720 and the width 90 This graph will occupy the majority of the screen, located approximately centrally.

The eight statements, starting with line 9 establish the scale factor, number of scaling divisions on the axes and the increment between data points. This follows the procedure outlined earlier in this chapter.

In line 17, the axes are drawn using these values. The final two parameters in the **xyaxes(..)** statement define the style of axes and colour. Here we have used literal value 0 and 3, rather than variables.

The seven statements beginning with line 18 generate and print the y axis labels, using th units calculated in the first part of the program. The string of labels is built up from the numbers required to be shown against the y axis. When the complete string has been generated, in this case containing 13 labels, it is drawn against the y axis using the **labely(..)** function.

Note that the vertical column of labels is offset 25 to the left of the origin of the axes to clear of the tick marks, and shifted down 25 units to centre each label against its y axis ti The 0+16 parameter right justifies the labels.

After the data file, BASICXY is opened, we prepare to load data by clearing the *dGE* d buffer with **datareset()**.

The **do..while** loop, starting at line 29, performs the dual function of reading data sequentially for each point, in this case at monthly intervals, and also deriving the x axis labels from the MONTH field in the database. Within the loop, the **datastore(..)** functi loads each data point, suitably scaled, into the data array.

The string variable, `Xlabels`, is built from the MONTH field, and displayed after all the data is loaded, using the **labelx**(..) function in line 34. The horizontal line of x labels is lowered 100 units to place it well below the x axis. The 0+8 parameter centres each label below the appropriate tick.

Drawing the the graph is now comparatively easy. The **xygraph**(..) function, line 35, displays it with style 2 and colour 5.

As we want to overlay some statistical information, we must first define the window within which these lines must lie. The **setwin**(..) function uses the dimensions of the graph to define the window.

The **stats**(..) and **bestfit**(..) functions can now be called. All the information they need to act on is in *dGE*'s memory. The data is in the internal array, the location of the graph has been defined by **xygraph**(..) and the window has been defined by **setwin**(..).

Note the relevance of the **setwin**(..) function. By expanding the window it is possible to project a best-fit line outside the immediate area of a graph. This process is called extrapolation and it allows you to view a trend projected into the future.

Before the final commands are executed we wait for a character to be typed by the user. If it is P the screen is printed on an attached laser printer, using the **prinipcl**(..) command. Following this we reset to text mode and quit the program.

To a large degree we have achieved in this program an automated graph drawing procedure. It is only a small step to make it auto-scaling. If we wish to change the scale of the graph to account for more sales next year, we have only to alter `MaxData`.

```
MaxData = 80000
```

It would be comparatively easy to test the data for the maximum value and adjust the value accordingly, with due allowance for the benefit of making it a round number. The `DataUnit` can be linked to it to produce sensible y axis increments.

If you wish to experiment with this program you will find it included in `SKELETON.PRG` on the issue disk.

Summary of a simple scaling operation

- Decide the required height of the graph in screen units.

  Required Height = 800 (4/5 the height of the screen)

- Decide the position of the graph on screen.

  Y origin = 100 (placing the axis in mid-screen)

- Decide the maximum value of the data you wish to portray.

  Maximum Data = 60000 (dollars)

- Decide the Data Unit and hence number of divisions on the axis.

  Data Unit = 5000 hence Number of divisions = 60000/5000 = 12

- Calculate the scale factor.

  Scale Factor = 800/60000 = 0.0133

- Follow similar procedures for the X axis.

Now -

- Draw the axes using the **xyaxes(..)** function.
- Transmit the data, multiplied by the scale factor.
- Draw the graph using the **xygraph(..)** function, (or **bargraph(..)** etc.)

Chapter 10

Problem solving

Text output to the graphics screen

Graphics modes do not support normal text output from programs. After you have set the screen to graphics using the **sethires**(..) function any subsequent text output will either be invisible or corrupt the screen in a random fashion. This includes output from the host program itself by way of the **status**, **talk** and other text based facilities which must be set **off**.

Host program error messages cannot be suppressed. The `Quit/Abort` style of message will often appear as a blur at the top of the screen, although this will depend on the graphics card you are using.

To assist in debugging programs, *dGE* traps inadvertent text ouput and automatically resets the screen to text mode. However this only works when the host program directs its text output via formal operating system calls.

Note the following differences between host programs.

dBASE III

Text mode is always restored when syntax errors occur.

Clipper

Clipper normally writes text directly to video memory. While developing your *dGE* application link the *Clipper* ANSI driver to your main program:

```
PLINK86 FI Yourprog FI ANSI.OBJ..etc ↵
```

and include `ANSI.SYS` in your `CONFIG.SYS` file.

```
device = ansi.sys
```

dGE is now able to trap error messages `Q/A/I` and display them at the top of the screen. Always select Q to quit.

Alternatively, in the Summer '87 compiler, you can execute your own UDF on the occurrence of an error. Place a **settext()** in this function.

FoxBASE+

FoxBASE+ normally writes text directly to video memory. While developing your application invoke *FoxBASE+* with the -NOTIBM switch:

```
FOXPLUS -NOTIBM Yourprog ↵
```

dGE is now able to trap error messages `C/I/S` and display them appropriately. Always select C to cancel.

C

Text mode is automatically restored if there is any text output from DOS error messages or *C* functions such as `printf`.

No graphics visible

If you see a flashing character appear on the left of the screen in place of graphics you have forgotten to run SETDGE.

If graphic pictures appear but break up you have probably allowed text output to the screen. A common error is to leave **status on** in *dBASE III*. The picture may break up when a data file is opened. Likewise **scoreboard** must be turned off.

No characters visible

A character set must first be loaded using the **loadcset**(..) function prior to using any of the text functions.

If ? appears in place of characters it is because a character set has not been loaded.

Check that **loadcset**(..) has the correct parameters and paths to find your character set.

Error reading data from dGE

The functions **get***something*(..) and **edstring**(..) functions use the *type ahead* keyboard buffer of your host program. You may find that you are unable to read the complete string as edited on screen.

To edit more than 16 characters **set typeahead to** a number larger than your maximum string length. (*Clipper* prior to Summer '87 did not support this function.)

In *dBASE* you are advised to flush the type ahead buffer prior to calling the **get***something*(..) functions. Also **set console off** before the **accept to** statement. Be sure to set it on afterwards or *dGE* will produce no output.

Too much data in a graph

The data array used in the **datastore**(..) function must be cleared after each graph, using the **datareset**() function. Failure to do so will result in data from the previous graph appearing on a subsequent one.

Rounding errors

dGE uses integer parameters in the range -32,768 to 32,767. Your host program deals in floating point numbers. When a floating point number is converted to an integer you lose the precision of any fractional part. In the majority of *dGE* applications this loss of precision is not discernable, being smaller than the resolution of the graphics screen.

A rounding error may become noticeable in the class of functions that involve *relative* moves and draws (vector and cartesian). A large number of relative moves may lead to a significant cumulative rounding error.

For this reason numbers are rounded correctly to the nearest integer within the interface library for those functions which involve relative geometry. Rounding may be introduced in all functions to increase precision. There will be an overhead, of course, in terms of execution time.

Note that in *dBASE* there is no interface library and you are obliged to **round(value,0)** within the parameter list of the calling function.

Chapter 11

dGE functions by classification

Mode selection functions

sethires	Set graphics mode
settext	Reset to text mode
setver	Set *dGE* version (V1 or V2 mode)

Cartesian drawing functions

boxfill	Draw a box and fill with pattern
drawcircle	Draw a circle or arc with radii and fill
drawline	Draw a line between two nominated points
drawxy	Draw a line from current X-Y position (relative)
movexy	Move current X-Y position (relative)
polyline	Draw a polyline figure

Vector drawing functions

drawvec	Draw a vector from current position (relative)
fixpos	Set current vector and X-Y position (absolute)
getcurx	Get current X position
getcury	Get current Y position
movevec	Move current vector position (relative)
polyvec	Draw a polyvector or polyline.

Clear functions

clrline	Clear a line of text
clrscreen	Clear the screen
clrstring	Clear the preceding string
clrwin	Clear a window

Text and character functions

edstring	Edit a string on-screen
getchar	Read a character from the keyboard
getstring	Read a string from the keyboard
labelpie	Draw Pie chart labels
labelx	Draw X axis labels
labely	Draw Y axis labels
loadcset	Load a character set
plotcset	Select plotter character set
suyicon	Print a symbol on the screen
suystring	Print a string on the screen
setdelim	Define delimiter for **edstring** and **getstring**
vecicon	Draw an icon (relative)
vecstring	Draw a character string (relative)

Data array functions

datapc	Define data % scale factor
datarange	Define a range of data to draw
datareset	Clear the data buffer
datastore	Store data for one data point
timedata	Add a data point to a time series graph

Data graphing functions

bargraph	Draw a bar graph
hlcgraph	Draw a High-Low-Close graph
piechart	Draw a pie chart
polaraxes	Draw polar axes
polargraph	Draw a polar graph
polyline	Draw a polyline figure
polyvec	Draw a polyvector or polyline
timegraph	Start a time series graph
xyaxes	Draw cartesian (X-Y) axes
xygraph	Draw a cartesian (X-Y) graph

Statistics functions

bestfit	Draw a best-fit line
getcc	Return the regression correlation coefficient
getmax	Return the maximum data value
getmean	Return the mean data value
getmin	Return the minimum data value
getsd	Return the standard deviation
minmax	Draw minimum and maximum lines
setwin	Define the window for drawing statistics
stats	Draw mean and standard deviations

Hardcopy functions

plotcset	Select plotter character set
plotoff	Disable output to plotter
ploton	Enable output to plotter
plotpen	Define the plotter pen characteristics
printfile	Name a file for printer/plotter output
printpcl	Print the screen on a PCL (laser) device
printscrn	Print the screen on a matrix printer

Miscellaneous utilities

setpal	Set the colour palette
diskfile	Read/write image to disk
getasin	Return the arcsin/arccos of a number
getsin	Return the sin/cos of an angle
shade	Shade the interior of a bounded area

Chapter 12

dGE Function Reference

This chapter contains a description of the *dGE* functions.

bargraph

Draw a bar graph

Usage:

`datastore(` *Amp, Patt, Sign, Colour*`)`

`bargraph(` *X, Y, Inc, Mode, Group*`)`

Amp	Amplitude of the bar (positive or negative except stacked)	
Patt	*0-19*	Pattern code
	20	Pattern = solid black (i.e. clear)
	+64	No pattern, just outline
	+128	No outline, just patternn
Sign	Unused (*dGE Version 1* requirement)	
Colour	0-15	
X	X origin	
Y	Y origin	
Inc	Increment of the independent variable (X or Y)	
Mode	*0*	Simple
	1	Stacked
	2	Clustered
	+4	Draw bars horizontally
	+8	Eliminate the spacer between adjacent bars
Group	Group size 0 to 125	

Return value:

Zero

Description:

This function draws a bar graph in either simple, stacked or clustered format. The bars may extend vertically or horizontally.

In stacked and clustered format each data point is represented by a group of data which must be loaded in the **datastore(..)** statement sequentially.

dGE resolves the grouping when the function is executed.

A space of a full bar width is placed between adjacent bars or groups of bars. Optionally this may be eliminated.

With simple graphs the group size is ignored (assumed 1).

Example:

```
datastore( Sales, 2, 0, 4)
datastore( Costs, 3, 0, 5)
datastore( Sales, 2, 0, 4)
datastore( Costs, 3, 0, 5)
bargraph(100, 100, 50, 1, 2)
```

will draw a vertical bar graph with origin at 100, 100 and with 50 units between each data point.

The graph is drawn in stacked form. The group size is 2 and each data point shows the amplitudes of the variables, Sales and Costs.

Sales are drawn with fill pattern 2 and colour 4. Costs are drawn with fill pattern 3 and colour 5. By default there is a space of a full bar between points.

dBASE III:

`&bargraph` *X, Y, Inc, Mode, Group* `end`

See also:

labelx
labely
xyaxes

bestfit

Draw a linear best-fit line

Usage:

`bestfit(` *Wid, Ht, Lstyle, Lcolour*)

Wid	Width of window to show best-fit
Ht	Height of window
Lstyle	Line style
Lcolour	0-15

Return value:

Zero

Description:

This function draws a linear best-fit line through the data set previously portrayed using one of the graphing functions.

A *least-squares regression* of Y on X is used to calculate this line.

Lines are drawn in a window defined by the positioning of the immediately preceding graph function.

The bottom-left of the window is taken from the origin of the graph. The width is given by the *Wid* parameter. The height is given by the *Ht* parameter.

Typically the width and height will correspond with those of the accompanying axes.

If *Wid* is 0 the window is defined by the **setwin(..)** function which must previously have been called.

Use the **setwin(..)** function if your graph extends in the negative X or Y directions.

Example:

```
bestfit( 1000, 500, 2, 8)
```

will superimpose a best-fit line on the graph previously drawn on the screen.

The best-fit line is drawn in a window of width 1000, and height 500. The lower left corne of the window is located at the origin of the preceding graph. The line style is 2. The line colour is 8.

dBASE III:

`&bestfit,` *Wid, Ht, Lstyle, Lcolour,* `end`

See also:

getcc
setwin

boxfill

Draw a box

Usage:

`boxfill(` *X, Y, Wid, Ht, Patt, Colour* `)`

X	X of bottom left	
Y	Y of bottom left	
Wid	Width	
Ht	Height	
Patt	*0-19*	Pattern code
	20	Pattern = solid black (i.e. clear)
	+64	No pattern, just outline
	+128	No outline, just pattern
Colour	0-15	

Return value:

Zero

Description:

This function draws a vertical sided rectangular box at a given co-ordinate with given width and height.

By default the interior of the box is automatically cleared before filling with the pattern.

Pattern 20 is solid black. This has the effect of clearing the interior to black, (or the prevailing background colour).

Pattern +64 produces just the outline without clearing the interior or filling it with a pattern.

Pattern +128 clears the interior and fills it with a pattern, without drawing an outline.

It may be noted that pattern = 20+128 produces the same effect as the *dGE Version 1* **clrwindow(..)** function, which is now superseded by **clrwin(..)**.

Example:

```
boxfill( 0, 0, 100, 200, 5+128, 10)
```

will draw a rectangle of pattern 5, colour 10, located at 0,0, with width 100 and height 200.

The rectangle has no outline.

dBASE III:

`&boxfill`, *X, Y, Wid, Ht, Patt, Colour,* end

See also:

shade

clrline

Clear a line of text

Usage:

`clrline( X, Y, Nchars )`

X	X position of start of string to clear
Y	Y position
Nchars	Number of characters to clear

Return value:

Zero

Description:

This function clears a line of characters.

See **clrstring()** for an alternative means of clearing **characters.**

Example:

```
clrline( 10, 10, 20)
```

will clear a line of 20 characters starting at 10,10

dBASE III:

`&clrline, X, Y, Nchars, end`

See also:

clrstring

clrscreen

Clear screen

Usage:

```
clrscreen()
```

Return value:

Zero

Description:

This function **clears** the entire screen to black or the background colour.

The *dBASE* **clear** function must never be used in graphics mode.

Example:

```
clrscreen()
```

dBASE III:

```
&clrscreen
```

clrstring

Clear a string

Usage:

```
clrstring()
```

Return value:

Zero

Description:

This function clears the string previously drawn with a **saystring** or **vecstring** function. This function is included for convenience to clear transient messages and prompts.

Example:

```
clrstring()
```

dBASE III:

```
&clrstring
```

See also:

clrline

Usage:

`clrwin( X0, Y0, X1, Y1 )`

X0	bottom left X
Y0	bottom left Y
X1	Upper right X
Y1	Upper right Y

Return value:

Zero

Description:

This function clears a rectangular window to black or the current background colour.

The *dGE Version 1* function **clrwindow(..)**, which uses different parameters, is still supported.

Example:

```
clrwin( 200, 300, 1000, 1000)
```

will clear a rectangular window with bottom left at 200, 300 and top right at 1000, 1000.

dBASE III:

`&clrwin, X0, Y0, X1, Y1, end`

See also:

boxfill

datapc

Adjust the scale of data

Usage:

```
datapc( Pc)
```

Pc Percentage scale factor 1 to 10000

Return value:

Zero

Description:

This function adjusts the scale of data, stored using **datastore**(..) or **timedata**(..) for all functions that execute after the **datapc**(..) is invoked. The default value of 100 (i.e. multiplying by 1) is restored after a **datareset**().

Note that this function applies the modifying factor to the graphing function. The data in the array remains unchanged. Hence any subsequent **datapc**(..) applies the modifying factor to the original data.

datpc(50) followed by **datpc(50)** does not produce a modifying factor of 0.5 x 0.5. The second **datpc**(50) produces no change.

Example:

```
datapc( 70 )
```

will reduce the amplitude of further graphs, charts or polylines to 70% until a **datareset**() is issued.

dBASE III:

```
&datapc, Pc, end
```

See also:

datareset
datastore
timedata

datarange

Define a subset of a data set

Usage:

`datarange( First, Last )`

First	Index to the first data point to be portrayed
Last	Index to the last data point to be portrayed

Return value:

Zero

Description:

This function is used to define a subset of the complete data set stored using **datastore(..)**.

Any graphing functions issued thereafter will portray only that range of data that lies between the first and last points nominated by this function.

The range defaults to the complete data set following a **datareset()**.

Indexes are base 0.

Example:

```
datarange( 3, 8 )
```

will identify the range of data from the 4th to the 9th point inclusive for graphing.

dBASE III:

`&datarange, First, Last, end`

See also:

datareset
datastore

datareset

Reset the data arrays

Usage:

```
datareset()
```

Return value:

Zero

Description:

This function emptics the *dGE* internal data array, resetting all data counts, range pointers and scale modifiers.

This applies to data loaded using the **datastore(..)** and **timedata(..)** functions.

The data scale factor, set by **datapc(..)**, is reset to 100%.

The range of data, set by **datarange(..)**, is reset to include the entire future array of data.

Example:

```
datareset()
```

will reset the data array.

dBASE III:

```
&datareset
```

See also:

datastore
timedata

datastore

Store data

Usage:

```
datastore( P1, P2, P3, P4)
```

P1	Numeric parameter
P2	Numeric parameter
P3	Numeric parameter
P4	Numeric parameter

Return value:

Zero

Description:

This function stores four parameters, representing amplitude and attribute information, in the *dGE* internal array.

Each **datastore**(..) stores data for a single data point. The count of data in the array is automatically incremented within *dGE*.

The size of the array is limited to 260 data points. Exceeding this number will result in the last ones being discarded.

The interpretation of the parameters is function specific. Refer to the various graph, chart and polyline functions for the meaning of the *P1* - *P4*.

Example:

```
datastore( 10, 20, 30, 0)
```

will store the values 10, 20, 30, 0 in the array at the current data point.

dBASE III:

```
&datastore, P1, P2, P3, P4, end
```

See also:

datareset
datapc
datarange

diskfile

Save or restore a screen image

Usage:

```
diskfile( Mode, File )
```

Mode	*0*	Read image from a file
	1	Write image to a file
File	File in form Drv:\Path\Filename	

Return value:

Zero

Description:

This function saves or restores a complete screen image to/from a named disk file.

The format in which it is stored in a file is currently not a standard interchange format.

The EGA screen is stored in compressed format. Typically this will result in a saving of space of up to 70%.

Example:

```
diskfile( 1, "A:\IMAGES\PIC01.DOT")
```

will write the current image to the file PIC01.DOT.

dBASE III:

```
&diskfile, Mode, txt, File, end
```

Usage:

```
drawcircle( X, Y, Rad, Ang1, Ang2, Mode, Style, Colour )
```

X	X centre	
Y	Y centre	
Rad	Radius	
Ang1	Starting angle of arc (positive or negative)	
Ang2	Ending angle of arc (positive or negative)	
Mode	0, 1, 2, 3	Line mode
	+4	Draw connecting radii
	+8	Fill arc segment with pattern
Style		Line style if no fill Pattern index if filled (forces line style = 0)
Colour	0-15	

Return value:

Zero

Description:

This functions draws a circle or arc in the specified line mode, style and colour.

Arcs are drawn from the first angle to the second in an anti-clockwise direction. Optionally radii may be drawn from the centre to the end points.

Optionally the segment or circle may be filled with a pattern. If this is specified the line style is forced to be continuous and if it is not a complete circle radii are drawn to ensure that the interior is bounded.

In some circumstances the filling of a partial circle may not be complete. See the **shade(..)** function for a note on the limitation of filling bounded areas in this version of *dGE*.

drawcircle

Example:

```
drawcircle( 1000, 500, 250, 45, 135, 0+4, 1, 5)
```

will draw an arc centred at 1000, 500 with radius 250 from 45 to 135 degrees, and draw radii at its ends.

dBASE II:

&drawcircle, *X, Y, Rad, Ang1, Ang2, Mode, Style, Col,* end

FoxBASE+:

drawcirc(..)

See also:

polaraxes

Usage:

`drawline( X0, Y0, X1, Y1, Lmode, Lstyle, Lcolour )`

X0	X start of line	
Y0	Y start of line	
X1	X end of line	
Y1	Y end of line	
Lmode	*0*	Replace
	1	Or
	2	Xor
	3	Black
	+16	Continue from last final point
Lstyle	*= 0*	Solid
	> 0	Broken with interval of Lstyle pixels
	+128n	Line index (for plotters only)
Lcolour	0-15	

Return value:

Zero

Description:

This function draws a straight line between two points specified in (absolute) cartesian co-ordinates.

Optionally the line may be continued from the last final position of a **drawline**(..). In this case it is not necessary to specify *X0, Y0* which may be given as 0 for convenience.

When used with plotter output the internal line style of the plotter may be selected by the *Lstyle* parameter.

Here the value *n* refers to the line style index of the plotter which may be used to select various mark/space patterns. Refer to your plotter manual for details, (HP-GL command LT). If n=0 the default is style 2 (even mark/space). Style 0, a single dot, is not available.

drawline

Example:

```
drawline( 10, 10, 100, 100, 0, 1, 1)
drawline( 0, 0, 200, 300, 0+16, 1, 1)
```

will draw a line from 10, 10 to 100, 100 with broken style in colour 1 (blue).

Following this a line is drawn from 100, 100 to 200, 300 using the chaining ability of the function.

dBASE III:

```
&drawline, X0, Y0, X1, Y1, Lmode, Lstyle, Lcolour, end
```

See also:

drawxy

Usage:

drawvec(*Len, Ang, Update, Lmode, Lstyle, Lcolour*)

Len	Length of vector in screen units	
Ang	Angle of vector in degrees positive or negative	
Update	0	Current position is updated to the final
	1	Current position is unchanged
Lmode	Line mode	
Lstyle	Line style	
Lcolour	0-15	

Return Value:

Zero

Description:

This function draws a vector line from the current position in a nominated style and colour.

The current position in absolute co-ordinates is by default 0, 0 until a draw, move or position function is executed.

Optionally the current position may be updated by this function. In this case the current cursor becomes the final point of the line.

Example:

```
drawvec( 100, 45, 0, 1, 2, 3)
```

will draw a vector from the current position of length 100 at an angle 45 degrees. The current position will be updated to the final position of the draw.

drawvec

dBASE III:

&drawvec, *Len, Ang, Update, Lmode, Lstyle, Lcolour,* end

See also:

drawxy
fixpos
movevec

drawxy

Draw a line to a relative end point

Usage:

`drawxy ( Xrel, Yrel, Update, Lmode, Lstyle, Lcolour )`

Xrel	Relative X position	
Yrel	Relative Y position	
Update	*0*	Current position **is updated** to the final
	1	Current position **is unchanged**
Lmode	Line mode	
Lstyle	Line style	
Lcolour	0-15	

Return value:

Zero

Description:

This function draws a line from the current position to a final position which is determined by relative X-Y co- ordinates.

The current position in absolute co-ordinates is by default 0, 0 until a draw, move or position function is executed.

Optionally the current position may be updated by this function. In this case the current position becomes the final point of the line.

Example:

```
drawxy( 300,400, 0, 1, 2, 3)
```

will draw a line from the current position to a final position given by –

Current X + 300, Current Y + 400

The current position will be updated to the final position of the drawing operation.

drawxy

dBASE III:

`&drawxy,` *Xrel, Yrel, Update, Lmode, Lstyle, Lcolour,* end

See also:

drawline
fixpos
movexy

edstring

Edit a string of characters

Usage:

edstring(*X, Y, Cset, Colour, String*)

X	X position of start of string to edit
Y	Y position
Cset	Character set number, 0-1
Colour	0-15
String	String to be edited

Return value:

Edited string

Description:

This function portrays a string on screen and allows it to be edited.

If X is 0, the starting point for the string on screen is calculated from the final point of a preceding **saystring**(..) function call. This allows simple chaining of prompts and responses.

By default it is delimited on screen by colons. This character can be changed or eliminated using the **setdelim**(..) function.

This function does not work with the inverse character set.

edstring

Editing keys are as follows -

↵	Terminate entry
←	Move one character to left
Ctrl S	Move one character to left
→	Move one character to right
Ctrl D	Move one character to right
BkSp	Delete character to left of cursor
Del	Delete character under cursor
Ctrl G	Delete character under cursor
Ctrl V	Toggle insert
Ctrl Y	Delete line

Example:

```
edstring( 100, 200, 0, 5, "dGE")
```

will print the string "dGE" at 100, 200 in colour 5 using character set 0, and allow it to be edited.

dBASE III:

```
&edstring, X, Y, Cset, Colour, txt, String, end
accept to Result
```

See also:

getchar
getstring

Usage:

```
fixpos( X, Y )
```

X	X position – absolute
Y	Y position – absolute

Return value:

Description:

This function fixes the vector and X-Y *current position* in absolute X,Y co-ordinates.

Subsequent vector or X-Y relative moves and draws are from the current position.

The current position may be further updated by any such moves and draws.

On entering graphics mode, the current position is set to 0,0.

Example:

```
fixpos( 100, 500)
```

will fix the current position for vector or X-Y relative draws or moves at 100, 500

dBASE III:

```
&fixpos, X, Y, end
```

See also:

drawvec
drawxy
movevec
movexy

getasin

Return ArcSin or ArcCos

Usage:

`getasin(` *Value, SinCos*)

Value	Value (x 1000) in range -10000 to 10000	
SinCos	*0*	Return ArcSin
	1	Return ArcCos

Return value:

Arcsin in range -90 to +90 (degrees)

Arccos in range 0 to 180

Description:

This function returns the ArcSin or ArcCos of a value expressed in degrees.

The value must first be scaled x10000 and passed as an integer.

The result is accurate to 2 decimal places.

Example:

```
Result = getasin( 7071, 0 )
```

Returns with `Result` = 45.00 (degrees)

dBASE III:

```
&getasin, Value, Sincos, end
accept to Result
```

In *dBASE* `Result` must be converted to numeric.

See also:

getsin

getcc

Get the linear regression correlation coefficient

Usage:

```
getcc()
```

Return value:

The linear regression correlation coefficient, -1.0000 to +1.0000.

Description:

This function returns the correlation coefficient of the least-squares regression of Y over X of the current data set.

The value is calculated from the complete set of points within the data array unless modified by **datarange(..)**, in which case it applies to the selected subset.

Since the correlation coeffient is dependent on both X and Y it is affected by the scale factor applied to X and Y which should be the same for consistency.

Note that if data is fixed increment, the graphing function must first be called to establish the X increment.

Integer round off may introduce an error of up to 0.1% on a full scale graph.

Example:

```
Result = getcc()
```

Returns with `Result` = Correlation coefficient, e.g. 0.8765.

dBASE III:

```
&getcc
accept to Result
```

In *dBASE* `Result` must be converted to numeric.

See also:

bestfit

getchar

Get a character from the keyboard

Usage:

```
getchar( Mode )
```

Mode	*0*	Wait for character from keyboard
	1	Sample for character
	2	Return the driver ID number
		'3' HERCULES
		'4' CGA mono
		'5' CGA colour
		'6' EGA
	+4	Convert lowercase to uppercase

Return value:

Character

Description:

This function returns a character from the keyboard either waiting or sampling for one.

Alternatively it returns the driver identifying character.

This function which was introduced for *dBASE II* has been superseded by the **Inkey()** function in *dBASE III* and compilers.

Example:

```
Character = getchar(0)
```

will wait for a character from the keyboard.

dBASE III:

```
&getchar, Mode, end
accept to Character
```

C:

```
getchr(Mode )
```

See also:

edstring
getstring

getcurx

Get current X position

Usage:

```
getcurx()
```

Return value:

Current X position in screen units (0 to 1350)

Description:

This function returns the current X position in (absolute) screen units, thereby enabling you to read the current screen position after vector moves and draws.

Example:

```
Xpos = getcurx()
```

Returns with Xpos in range 0 to 1350

dBASE III:

```
&getcurx
accept to Xpos
```

In *dBASE* Xpos must be converted to numeric.

See also:

getcury

getcury

Get current Y position

Usage:

```
getcury()
```

Return value:

Current Y position in screen units (0 to 1000)

Description:

This function returns the current Y position in (absolute) screen units, thereby enabling you to read the current screen position after vector moves and draws.

Example:

```
Ypos = getcury()
```

Returns with Ypos in range 0 to 1000

dBASE III:

```
&getcury
accept to Ypos
```

In *dBASE* Ypos must be converted to numeric.

See also:

getcurx

getmax

Get maximum value of a data set

Usage:

```
getmax()
```

Return value:

Maximum value of the data set

Description:

This function returns the maximum value of data in the current dataset as stored using **datastore(..)**.

The value in screen units is calculated from the complete data set unless reduced to a subset by **datarange(..)**.

The value calculated is also modified by any prevailing **datapc(..)** function

In other words, the value returned in screen units relates to the data as would be currently viewed in any of the graphing functions.

To restore this value to your true data value you should divide by any modifying factor used in the **datastore(..)** or **datapc(..)** functions.

While it is accepted that this value is less precise than that which may be obtained from the original data, (owing to integer round off), this function has a useful application in scaling and positioning on screen.

Example:

```
Result = getmax()
```

Returns with `Result` = Maximum value in screen units.

dBASE III:

```
&getmax
accept to Result
```

In *dBASE* `Result` must be converted to numeric.

See also:

getmin
getmean
minmax

getmean

Get the mean value of a data set

Usage:

```
getmean()
```

Return value:

Mean value of the data set

Description:

This function returns the mean value of data in the current dataset as stored using **datastore(..)**.

This is the same value as drawn in the **stats(..)** function, although it should be noted that the **stats(..)** function does not have to be called previously.

The value in screen units is calculated from the complete data set unless reduced to a subset by **datarange(..)**.

The value calculated is also modified by any prevailing **datapc(..)** function

In other words, the value returned in screen units relates to the data as would be currently viewed in any of the graphing functions.

To restore this value to your true data value you should divide by any modifying factor used in the **datastore(..)** or **datapc(..)** functions.

While it is accepted that this value is less precise than that which may be obtained from the original data, (owing to integer round off), this function has a useful application in scaling and positioning on screen.

Example:

```
Result = getmean()
```

Returns with **Result** = Mean value in screen units.

```
accept to Result
```

In *dBASE* `Result` must be converted to numeric.

See also:

stats
getmin
getmax

getmin

Get the minimum value of a data set

Usage:

```
getmin()
```

Return value:

Minimum value in the data set

Description:

This function returns the minimum value of data in the current dataset as stored using **datastore**(..).

The value in screen units is calculated from the complete data set unless reduced to a subset by **datarange**(..).

The value calculated is also modified by any prevailing **datapc**(..) function

In other words, the value returned in screen units relates to the data as would be currently viewed in any of the graphing functions.

To restore this value to your true data value you should **divide** by any modifying factor used in the **datastore**(..) or **datapc**(..) functions.

While it is accepted that this value is less precise than that which may be obtained from the original data, (owing to integer round off), this function has a useful application in scaling and positioning on screen.

Example:

```
Result = getmin()
```

Returns with `Result` = Minimum value in screen units.

dBASE III:

```
&getmin
accept to Result
```

In *dBASE* `Result` must be converted to numeric.

See also:

getmax
getmean
minmax

getsd

Get standard deviation of a data set

Usage:

```
getsd()
```

Return value:

Standard deviation of the data set

Description:

This function returns the standard deviation of the current dataset as stored using **datastore(..)**.

This is the same value as drawn in the **stats(..)** function, although it should be noted that the **stats(..)** function does not have to be called previously.

The value in screen units is calculated from the complete data set unless reduced to a subset by **datarange(..)**.

The value calculated is also modified by any prevailing **datapc(..)** function

In other words, the value returned in screen units relates to the data as would be currently viewed in any of the graphing functions.

To restore this value to your true data value you should divide by any modifying factor used in the **datastore(..)** or **datapc(..)** functions.

Integer round off may introduce an error of up to 0.1% on a full scale graph.

Example:

```
Result = getsd()
```

Returns with **Result** = Standard deviation in screen units.

dBASE III:

```
&getsd
accept to Result
```

In *dBASE* **Result** must be converted to numeric.

See also:

stats
getmean

getsin

Get sine or cosine

Usage:

`getsin( Ang, SinCos )`

Ang	Angle in degrees (positive or negative)	
SinCos	*0*	Return Sin
	1	Return Cos

Return value:

Value to four significant figures

Description:

This function returns the sine or cosine of an angle.

Example:

```
Result = getsin( 45, 0 )
```

Returns with `Result` = 0.7071

dBASE III:

```
&getsin, Ang, Sincos, end
accept to Result
```

In *dBASE* `Result` must be converted to numeric.

See also:

getasin

Usage:

`getstring(` *X, Y, Cset, Nchars, Colour* `)`

X	X position of string to read
Y	Y position
Cset	Character set number 0 or 1
Nchars	Number of characters to read
Colour	Colour of echoed characters, 0-15

Return value:

Description:

This function reads a string entered at the keyboard and echoes it on screen. The string is terminated by hitting ↵.

If X is 0 the starting point for the string on screen is calculated from the final point of a preceding **saystring**(..) function call. This allows simple chaining of prompts and responses.

By default it is delimited on screen by colons. This character can be changed or supressed using the **setdelim**(..) function.

This function does not work with the inverse character set.

See **edstring**(..) for an explanation of the editing control keys.

Example:

```
Result = getstring(100, 200, 1, 10, 5)
```

will prompt for a 10 character string to be entered at 100, 200 using character set 1 in colour 5.

getstring

dBASE III:

```
&getstring, X, Y, Cset, Nchars, Colour, end
accept to Result
```

See also:

edstring
getchar

Usage:

`datastore(` *A, B, C, Xpos*)

`hlcgraph(` *X0, Y0, Xinc, Colour*)

A	High	
B	Low	... in any order
C	Close	
Xpos	X position of data point in scatter format	
X0	X origin of graph	
Y0	Y origin of graph	
Xinc	> *0*	Data point increment in X
	= *0*	The data is drawn in scatter format
Colour	Colour of HLC symbols, 0-15	

Return value:

Zero

Description:

This function draws a high-low-close graph using data from the *dGE* array.

The X position may either be at fixed increments or at arbitrary (scatter) positions.

The parameters in the **datastore(..)** function may be in any order of High-Low-Close. The graphing function sorts them into descending rank. If you do not want to plot a *Close* value you must still provide a value for it. This must be in the range *Low* to *High* and could be, for example, *Low*, *High* or the mean of these. The latter would allow you to use the statistics functions to analyse the mean value.

Statistics functions may be overlaid. These functions use the first parameter in the **datastore(..)** function. Hence by ordering the **datastore(..)** parameters it is possible to select the field (High, Low, or Close) that the statistics functions act on.

hlcgraph

Example:

```
datastore( 30, 50, 100, 70)
hlcgraph( 100, 100, 0, 3 )
```

will draw an HLC graph with origin at 100, 100 and colour index 3

The descending values 100, 50, 30 are drawn in scatter format at X = 70.

If statistics lines are added to the above example, they are calculated from the low parameter value, as it is the first parameter of the **datastore(..)** function. This, of course, should be consistent in all the **datastore**'s.

dBASE III:

```
&hlcgraph, X0, Y0, Xinc, Colour, end
```

See also:

labelx
labely
xyaxes

Usage:

`labelpie(` *Xoff, Rad, LabLen, Cset, Mode, Colour, String* `)`

Xoff	X offset of arc for drawing labels	
Rad	Radius of arc for drawing labels	
LabLen	Length of each label (number of characters)	
Cset	Character set number 0 or 1	
Mode	+2 don't clear (i.e. superimpose)	
Colour	*0*	use segment colours
	1-15	all labels this colour
String	Compound string of labels (1 to 255 chars long)	

Return value:

Zero

Description:

This function draws a sequence of text labels to complement a pie chart. The pie chart must be drawn first since this function adopts certain parameters from the preceding **piechart**(..) function call.

The labels must be packed in a single string e.g.. `'JanFebMarApr'`.

Each label is of fixed length, *LabLen*, characters long.

There must be the same number of labels as pie elements.

The angular position of each label is calculated from the data **datastore**'d for the pie chart.

The labels are drawn in an arc on each side of the pie, connected to their respective segments by *pointing* lines.

These lines are drawn from the label horizontally a distance *Xoff*, then radially towards the centre of the pie, in a direction bisecting the segment.

labelpie

The radius of the arc on which the labels are drawn is determined by *Rad*. THIS MUST BE GREATER THAN THE PIE RADIUS, which is taken from the preceding **piechart(..)** call.

It is recommended that *Rad* is greater tha 1.1 times the pie radius, (or 1.35 times the radius if any segments are "exploded").

The colour of the labels may either correspond to those of their respective pie segments (*Colour* = 0) or be uniformly the same (*Colour* > 0)

The function is not able to prevent labels over printing if the pie segments are too close. In this event it may be preferrable to select the *superimpose* mode.

The separation between labels is a function of *Rad*. It is greatest when *Rad* is just less than 500 with the pie positioned in the centre of the screen.

Example:

```
Labels = 'JanFebMar'
piechart(675, 500, 300)
labelpie(200, 450, 3, 0, 2, 7, Labels)
```

will draw three labels around a pie chart centred at 675, 500. The arc on which labels are drawn has a radius of 450, displaced 200 units to either side of the centre of the pie. (The centre of the arcs are at X = 675-200 and X = 675+200).

The text uses character set 0, with colour 7 and will superimpose.

dBASE III:

`&labelpie(` *Xoff, Rad, LabLen, Cset, Mode, Col, String*`)` end

See also:

piechart

Usage:

`labelx( X, Y, Xinc, LabLen, Cset, Mode, Colour, String )`

X	X position of first label	
Y	Y position of first label	
Xinc	Distance between labels in X direction	
LabLen	Length of each label (number of characters)	
Cset	Character set number 0 or 1	
Mode	*0*	draw horizontally
	1	draw vertically
	+2	don't clear (i.e. superimpose)
	+8	position my middle
	+16	position by end
Colour	0-15	
String	Compound string of labels (1 to 255 chars long)	

Return value:

Zero

Description:

This function draws a horizontal sequence of text labels starting on screen at *X,Y* and at intervals of *Xinc* to the right of this point.

The labels must be packed in a single string, e.g. `'JanFebMarApr'`.

Each label is of fixed length, *LabLen*, characters long.

The number of labels drawn is equal to the length of the string divided by the length of the label. *dGE* calculates this automatically.

The choice of character set, positional and drawing modes are the same as **saystring(..)** except that the *descending mode* is ignored.

Note that numeric labels may easily be built using the **str()** function in a **do..while** loop.

labelx

Example:

```
Label = 'JanFebMar'
labelx(100, 100, 150, 3, 1, 0+8, 2, Label)
```

will draw three labels in a row starting from the point 100, 100 at intervals of 150 units in the X direction. The labels are centre justified and are drawn in colour 2 using character set 1.

```
Val = 0
Label = ''
do while Val < 10
  Label = Label + STR(Val, 4, 1)
  Val = Val+0.5
enddo
```

will pack 20 labels of length 4 into the string `Label`.

Note that the Label string may also be built from a text field in a database (e.g. names, countries, time), but DATE types must first be converted to character using **dtoc()**.

dBASE III:

`&labelx`, *X, Y, Xinc, LabLen, Cset, Mode, Colour,* `txt`, *String,* `end`

See also:

labely
xygraph
hlcgraph
bargraph

Usage:

`labely(` *X, Y, Yinc, LabLen, Cset, Mode, Colour, Smng*`)`

X	X position of first label	
Y	Y position of first label	
Yinc	Distance between labels in Y direction	
LabLen	Length of each label (number of characters)	
Cset	Character set number 0 or 1	
Mode	*0*	draw horizontally
	1	draw vertically
	+2	don't clear (i.e. superimpose)
	+8	position my middle
	+16	position by end
Colour	0-15	
Smng	Compound string of labels (1 to 255 chars long)	

Return value:

Zero

Description:

This function draws a vertical sequence of text labels starting on screen at *X,Y* and at further intervals of *Yinc* above this point.

The labels must be packed in a single string, e.g. `'JanFebMarApr'`.

Each label is of fixed length, *LabLen*, characters long.

The number of labels drawn is equal to the length of the string divided by the length of the label. *dGE* calculates this automatically.

The choice of character set, positional and drawing modes are the same as **saystring**(..) except that the *descending mode* is ignored.

labely

Example:

See **labelx**(..)

dBASE III:

`&labely`, *X, Y, Yinc, LabLen, Cset, Mode, Colour,* `txt`, *String,* `end`

See also:

labelx
xygraph
hicgraph
bargraph

loadcset

Load a character set file

Usage:

`loadcset( Cset, File )`

Cset Character set number 0 or 1.
File File in form `Drv:\Path\Filename` where `Filename` assumes the extension `.CHR`

Return value:

Zero

Description:

This function loads a named character set from a file on disk into one of the two current character buffers.

A character set must be loaded before any of the string functions are called. Failure to do so will result in all characters appearing as ?.

Example:

```
loadcset( 0, "C:\CSETS\STANDARD")
```

will load the character set STANDARD.CHR into buffer 0.

dBASE III:

`&loadcset, Cset, txt, File, end`

minmax

Draw minimum and maximum lines

Usage:

`minmax(` *Wid, Lstyle, Lcolour* `)`

Wid	Width of window	
Lstyle	*= 0*	Solid
	> 0	Broken with interval of Lstyle pixels
	+128n	Line index (for plotters only)
Lcolour	0-15	

Return value:

Zero

Description:

This function draws lines representing the minimum and maximum of the data set previously transmitted using **datastore**(..) and shown with one of the graph functions.

The values on which it acts are those of the first parameter in the **datastore(..)**, i.e. *P1*. This has relevance to the High-low-close function.

Lines are drawn in a window defined by the positioning of the immediately preceding graph function.

The bottom-left of the window is taken from the origin of the graph. The width is given by the *Wid* parameter.

If *Wid* is 0 the window is defined by the **setwin**(..) function which must previously have been issued.

Use the **setwin**(..) function if your graph extends in the negative X direction.

Example:

```
minmax( 100, 1, 2 )
```

will draw a minimum and maximum of the data set with line style 1 and colour 2.

The line will be drawn in a window of width 100 extending in the positive X direction from the origin of the preceding graph.

dBASE III:

`&minmax,` *Wid, Lstyle, Lcolour,* `end`

See also:

getmin
getmax

movexy

Move current position by X, Y offset

Usage:

`movexy ( Xrel, Yrel )`

Xrel	Relative X move
Yrel	Relative Y move

Return value:

Zero

Description:

This function moves the current position by a distance given by *Xrel, Yrel*.

The new position applies for the next vector or X-Y draw or move.

Example:

```
movexy( 200, -300 )
```

will move the current position by 200 screen units in the X direction and -300 units in the Y direction.

dBASE III:

`&movexy, Xrel, Yrel, end`

See also:

flxpos
drawxy

movevec

Move current position by vector length and angle

Usage:

```
movevec( Len, Ang )
```

Len Length of vector in screen units
Ang Angle of vector in degrees positive or negative

Return value:

Zero

Description:

This function moves the current position by a vector length and angle, (relative).

The new position applies for the next vector draw or move.

Example:

```
movevec( 100, 45 )
```

will move the current position by 100 screen units in a direction of 45 degrees (counter-clockwise from the X direction).

dBASE III:

```
&movevec, Len, Ang, end
```

See also:

fixpos
drawvec

piechart

Draw a piechart

Usage:

`datastore(` *Value, Patt, Expl, Colour* `)`

`piechart(` *X, Y, Rad* `)`

Value	Data value, positive integer (0 to 32768)
Patt	Segment fill pattern
Expl	*0* segment origin at centre *1* segment origin displaced (1/4 of the radius)
Colour	Segment colour, 0-15
X	X centre of pie
Y	Y centre of pie
Rad	Radius of pie, (> 25 units)

Return value:

Zero

Description:

This function draws a pie chart from accumulated data. Each segment may be individually patterned, coloured and "exploded".

The size of a segment is determined as a proportion of the total of **all** *Values* in the data set.

The *Value* itself does not have to be expressed as a percentage. *dGE* calculates the percentage automatically.

If your values are not percentages we suggest you scale them to 1000.

```
Factor = 1000 / MAXVALUE                    && estimate
r = datastore( Value * Factor, .. , ..
```

This allows a wide margin of error in the estimate of maximum value while retaining high accuracy (better than 0.1%).

Once drawn, the piechart may be labelled using the labelpie(..) function.

Example:

```
datastore(10, 1, 0, 3)
datastore(30, 2, 1, 4)
piechart(675, 500, 300)
```

will draw a pie chart with two segments centred at 675, 500 with radius 300.

The first segment is 25% of the whole and is pattern 1, colour 3.

The second segment is 75% of the whole and is pattern 2, colour 4 and is exploded.

dBASE III:

`&piechart` *X, Y, Rad,* `end`

See also:

labelpie

plotcset

Select plotter character set

Usage:

`plotcset(` *DgeCset, Height, Width, Font*)

DgeCset	**0 or 1** .. the *dGE* character set number
Height	**Height** of characters as a percentage of the standard *dGE* font. 100 = standard.
Width	**Width** of characters as a percentage of the standard *dGE* width. 100 = standard
Font	**Identifying** code for the plotter's International character sets

Return value:

Zero

Description:

This function defines characteristics of the plotter's internal font to be used in subsequent text operations. Two current character sets may be selected, 0 and 1. These correspond to the character set numbers within *dGE*.

The default height and width is 100 corresponding to the STANDARD *dGE* font.

When *dGE* draws characters on screen the different fonts allow you to change the size and style (SMALL, MEDIUM etc.) but not the character separation. Note that with plotters you may not change the style but you can change separation, using the *Width* parameter. If you want to be consistent with characters seen on screen always select a *Width* of 100.

The fourth parameter selects an international character set. Typically –

US	0
GERMANY	33
FRANCE	34
UK	35

Refer to your manual for details.

Example:

```
plotcset( 0, 60, 100, 1)
```

will select a plotter character set and style for the *dGE* character set 0.

Characters will be drawn 60% of the height of the *dGE* STANDARD character and the spacing of characters remains the same as on screen.

The plotter's international font 1 is selected.

dBASE III:

`&plotcset,` *DgeCset, Height, Width, PlotCset,* `end`

See also:

ploton
plotoff

plotoff

Terminate plotter output

Usage:

```
plotoff( )
```

Return value:

Zero

Description:

This function terminates plotter output to device or file. If output has been re-directed to disk the file is closed.

Example:

```
plotoff()
```

dBASE III:

```
&plotoff
```

See also:

ploton

Usage:

```
ploton( Hoff, Voff, Xlen, Mode, Units )
```

Hoff	Horizontal offset (mms) from left margin	
Voff	Vertical offset (mms) from base	
Xlen	Width of plot in X direction (mms)	
Mode	*0*	Horizontal (landscape)
	1	Vertical (portrait)
Units	Plotter units per mm.	

Return value:

Zero

Description:

This function defines initial plotter parameters and starts to re-direct graphical output to a plotter or plotter file.

Hoff and *Voff* define the position of the bottom left corner of the figure relative to the left side and base of the plotter table.

Xlen defines the length of the figure in the X direction (as viewed on the screen) and hence the overall size of the figure.

The aspect ratio as seen on the screen is maintained.

Units defines the unit of length for your particular plotter. For many A3 and A4 plotters the basic unit is 0.025 mm. Hence plotter units / mm = 40.

Fill patterns and characters may not conform to those seen on screen since the driver uses the internal fonts and styles of the plotter.

The driver adheres to the HP-GL code standard.

All dimensions are in millimetres.

ploton

Screen output may occur concurrently if **sethires**(..) is active.

Example:

```
ploton( 20, 50, 300, 0, 40)
```

will begin to draw a plot at 20, 50 mms offset from the lower left of the plot table. The width of the plot in the X direction is 300 mms and it is drawn in landscape style.

The plotter units are 40 per mm. (0.025 mm discrimination).

dBASE III:

```
&ploton, Hoff, Voff, Xlen, Mode, Units, end
```

See also:

plotcset
plotpen
plotoff

plotpen

Allocate pen numbers to colours

Usage:

`plotpen(` *Colour, PenNo, PenThick* `)`

Colour ***dGE*** colour index 1-15
PenNo **Number** of the plotter pen, 1 to plotter maximum
PenThick **Pen thickness** in units of 0.1 mm

Return value:

Zero

Description:

This function defines the relationship between the colours used by *dGE* on screen and the corresponding pens on your plotter.

Pen thickness is also required in order to produce proper "fills". It is defined in units of 0.1 mm. Hence a pen thickness of 0.3 requires a thickness parameter of 3.

The default pen for all colours is 1 with thickness 0.3mm. This will be assumed unless overridden by this command.

Example:

```
plotpen( 7, 1, 3)
```

assigns ***dGE*** colour 7 (white in colour versions) to pen number 1, usually black.

The pen thickness is 0.3 mm.

dBASE III:

`&plotpen,` *Colour, PenNo, PenThick,* `end`

plotpen

See also:

ploton
plotoff

Usage:

`polaraxes( X, Y, Rad, Ndivs, Colour )`

X	X centre
Y	Y centre
Rad	Radius of arms
Ndivs	Number of divisions per arm +128 .. draw circular grids
Colour	Colour of arms and grids, 0-15

Return value:

Zero

Description:

This functions draws a set of polar axes at 45 degree increments. Each arm is divided by ticks at regular intervals.

Optionally, circular grids may be superimposed at the same intervals.

Example:

```
polaraxes( 675, 500, 300, 10 + 128, 5)
```

will draw axes centred at 675, 500 with arm length 300 divided into 10 divisions. Circular grids are superimposed.

dBASE III:

`&polaraxes, X, Y, Rad, Ndivs, Colour, end`

polaraxes

See also:

polargraph

Usage:

```
datastore( Amp, Icon, Ang, 0 )
polargraph( X, Y, Cycles, Style, Colour )
```

Amp	Amplitude of the data point	
Icon	Icon code	
Ang	Angle (independent variable) in scatter format	
X	X origin	
Y	Y origin	
Cycles	*> 0*	Number of complete cycles of data
	= 0	Scatter format
Style	*0*	Chained
	1	Icons
	2	Chained and Icons
	3	Sticks
Colour	0-15	

Return value:

Zero

Description:

This functions draws a polar graph in either fixed increment or scatter format.

In fixed increment mode the angular increment is calculated from the complete number of cycles of data and the number of data points.

One cycle of data is represented in 360 degrees. Further cycles repeat, 360 to 720 etc

Note that different icons can be attached to data points to distinguish repeating cycles.

In scatter format the angle is taken from *P3* in the **datastore(..)**.

polargraph

Example:

```
datastore( 100, 1, 45, 0 )
datastore( 200, 1, 90, 0 )
datastore( 300, 1,135, 0)
polargraph( 675, 500, 0, 0, 8)
```

will draw a "chained" polar graph centred at 675, 500 in scatter format and colour 8.

The vectors defining the data points are 100, 45; 200, 90; 300, 135.

dBASE III:

`&polargraph` *X, Y, Cycles, Style, Colour,* `end`

FoxBASE+:

`polargr(..)`

See also:

polaraxes

polyline

Draw a figure of connected lines

Usage:

`datastore(` *Xrel, Yrel, 0, 0* `)`

`polyline(` *X, Y, Lmode, Lstyle, Lcolour* `)`

Xrel	X position of this point relative to origin
Yrel	Y position of this point relative to origin
X	X origin of figure
Y	Y origin of figure
Lmode	Line mode
Lstyle	Line style
Lcolour	0-15

Return value:

Zero

Description:

This function draws a figure of connected lines as defined in the previous **datastore(..)** calls.

The figure is located arbitrarily on screen by X and Y.

The function has been superseded in all respects by **polyvec(..)**.

dBASE III:

`&polyline,` *X, Y, Lmode, Lstyle, Lcolour,* `end`

See also:

polyvec

polyvec

Draw a figure of connected lines

Usage:

`datastore(` *A, B, Move, 0*`)`

`polyvec(` *Pos1, Pos2, Mode, Ang, Lmode, Lstyle, Lcolour* `)`

A	X in cartesian mode Length in vector mode	
B	Y in cartesian mode Angle in vector mode	
Move	*0*	Draw the line to this point
	1	Just move the current position
Pos1	X origin of figure /or Length of vector	
Pos2	Y origin of figure /or Angle of vector	
Mode	*0*	Origin is in X,Y format (cartesian)
	1	Origin is in Len, Angle format (vector)
	+0	Data is in X,Y format
	+2	Data is in Len, Angle format
	+4	Delete the last draw before re-drawing
	+8	Mirror about vertical
	+16	Mirror about horizontal
Ang	Angle of rotation about the origin, positive counter-clockwise	
Lmode	Line mode	
Lstyle	Line style	
Lcolour	0-15	

Return value:

Zero

Description:

This function draws a figure of connected points as defined in the **datastore** array.

The locating position (origin) may either be in X,Y format or vector Length, Angle format relative to the current position.

The values in the array may either be in X,Y co-ordinates or vector Length, Angle. These are relative to the origin of the figure set by *Pos1, Pos2*.

The whole figure may be rotated about the origin in a counter-clockwise direction.

The figure may be mirrored about the vertical axis, or the horizontal axis or both.

Mirroring is performed before rotation, i.e. it mirrors the **datastore**'d figure **not** the drawn figure.

In addition to technical drawing, this function enables you to define symbols **and** characters. Using the **datarange**(..) function you can index into a bank **of figures** stored in **sequence**.

Note that this function allows two levels of relativity. The elements *(A,B)*, **defined** by the **datastore**(..) function, are relative to the origin of the figure.

The origin, *(Pos1,Pos2)*, defined in the **polyvec**(..) function, is relative to **the** *current position*, as defined by **flxpos**(..) and any subsequent **movexy**(..) or **movevec**(..).

Issue **flxpos**(0,0) prior to **polyvec**(..) to make *(Pos1,Pos2)* absolute positions.

Example:

```
polyvec( 100, 200, 0+4, 45, 1, 2, 3)
```

will draw a figure from the **datastore**'d array located at 100, 200 relative to the current position and rotated through 45 degrees. The previous figure drawn with this function is deleted prior to re-drawing.

dBASE III:

`&polyvec,` *Pos1, Pos2, Mode, Ang, Lmode, Lstyle, Lcol,) end*

polyvec

See also:

fixpos

Re-direct printer/plotter output to a file

Usage:

```
printfile( Mode, File )
```

Mode	*0*	Disable output to file
	1	Enable output
	+2	Append to existing file. (Default is to overwrite)
File	`Drv:\Path\File.ext`	

Return value:

Zero

Description:

This function defines a file name for printer or plotter output and enables output whenever a printer or plotter function is called later.

The file is opened by the **printscrn()**, **printpcl(..)** and **ploton(..)** functions.

If it does not exist it is created. If it does exist it is overwritten unless the *append* mode is specified.

The file is closed by the **plotoff()** function or on leaving the **printscrn(..)** and **printpcl(..)** functions

It may be printed or spooled as if it were a text file.

Note that raster image files must be designated as BINARY when using the DOS COPY command.

The DOS print spooler, PRINT, does not accept binary files. It terminates when it encounters a CTRL Z character.

printfile

Example:

```
printfile( 1, "A:\PRINT\PICTURE.PRN")
```

will direct any subsequent raster or plotter printing into a file.

```
printfile( 0, "")
```

will disable re-direction.

```
copy picture.prn lpt1: /b↵
```

at the DOS prompt will copy the file to the printer.

dBASE III:

```
&printfile, Mode, txt, File, end
```

See also:

printscrn
printpcl
ploton
plotoff

printpcl

Output to a laser printer

Usage:

`printpcl` (*Mode, Hoffset, Voffset, Density*)

Mode	+1	Issue reset before printing
	+2	Issue form-feed after printing
	+4	Correct the aspect ratio
Hoffset	Horizontal offset on page in dots	
Voffset	Vertical offset on page in dots	
Density	Dot density 75, 100, 150, 300 (per inch)	

Return value:

Zero

Description:

This function defines printing parameters and starts to direct raster graphics printing to a laser printer or file.

The offsets are specified from the top of the page and from the left margin. These are in units of dots. There are 300 dots per inch.

The printer driver conforms to the HEWLETT-PACKARD Printer Command Language (PCL) as implemented on the HP Laserjet.

On completion an open print file is closed.

The screen aspect ratio is 1:1.35. However, the density of dots on screen is not uniform – the horizontal density generallly exceeds the vertical density. As a result a direct "dump" of the raster image to a laser printer, which has uniform density, will produce an elongated image. This distortion is most apparent with circles.

The printpcl(..) function has the option to correct the aspect ratio by *stretching* the vertical axis to produce an image with ratio 1:1.35.

The side effect is that every so often a line or row is doubled up, which can lead to unpleasant interference effects with regular patterns.

printpcl

Example:

```
printpcl( 2, 100, 200, 150 )
```

will print the screen on a laser printer. The figure is offset 100 dots from the left margin and 200 dots from the top of the page.

It is printed with a dot density of 150 dots/inch.

After printing a form-feed is sent.

The picture is not corrected for aspect ratio.

dBASE III:

`&printpcl,` *Mode, Hoffset, Voffset, Density,* `end`

See also:

printfile

printscrn

Output to a matrix printer

Usage:

```
printscrn( )
```

Return value:

Zero

Description:

This function prints the screen using EPSON-style control codes.

The output can go straight to an attached printer or be directed to a print file using the **printfile(..)** command.

The control codes can be customised using the CONFIGP utility.

Example:

```
printscrn()
```

dBASE III:

```
&printscrn
```

See also:

printfile

sayicon

Draw a symbol

Usage:

```
sayicon( X, Y, Mode, IconId, Colour )
```

X	X position of centre of symbol	
Y	Y position	
Mode	*0*	User defined icon
	1	Library icon
IconId	*Mode 0*	32 character string
	Mode 1	0-9 library code
Colour	0-15	

Return value:

Zero

Description:

This function draws a symbol on the screen which can be selected from the internal library or be defined by the user.

If it is user defined, a 32 character string is required to define its shape.

It is composed of a 16 x 16 bit cell defined by the binary ASCII value of each character. There are 16 rows of two bytes ordered left to right and top to bottom. The least significant bit is on the left.

Example:

```
sayicon( 10, 20, 1, 8, 9)
```

will draw library symbol 8 at 10, 20 in colour 9.

sayicon

dBASE III:

&sayicon, *X, Y, Mode, Colour,* txt, *IconId,* end

Or

&sayicon, *X, Y, Mode, Colour, IconId,* end

Note the re-ordering of the *Colour* and *IconId* parameters.

See also:

vecicon

saystring

Display a string of text

Usage:

`saystring(` *X, Y, Cset, Mode, Colour, String*)

X	X position of string	
Y	Y position	
Cset	Character set number 0 or 1	
Mode	*0*	draw horizontally
	1	draw vertically
	+2	don't clear (i.e. superimpose)
	+4	descend from previous
	+8	position my middle
	+16	position by end
Colour	0-15	
String	Text string	

Return value:

Zero

Description:

This function draws a text string at X,Y either horizontally or vertically (with descending characters), using one of the two current character sets.

Optionally the line is not cleared first, (i.e. it is superimposed over another image).

Optionally the string may be positioned on the line immediately below the previous string. In this case no Y value is required. However, the X value is still required to determine its horizontal position. This feature is not relevant to vertical strings.

By default the string is located by its bottom left corner.

Optionally it may be positioned by its centre or by the rightmost character.

Example:

```
saystring( 100, 200, 1, 0+2+16, 5, "TEST STRING")
```

will draw the string TEST STRING using character set 1.

The string is located at 100, 200 by the lower right corner of the character "G".

The line is not cleared before superimposing the characters.

dBASE III:

`&saystring, X, Y, Cset, Mode, Colour, txt, String ,end`

See also:

vecstring

tdelim

ine prompt string delimiter

age:

```
setdelim( AscChar )
```

AscChar The ASCII value of the delimiting character

eturn value:

Zero

)escription:

This function defines the character to be shown at the beginning and end of **getstring(..)** and **edstring(..)** prompts.

If *AscChar* = 0 no delimiter is used.

Example:

```
setdelim( 65 )
```

will set the character A as the delimiter.

dBASE III:

```
&setdelim, AscChar, end
```

See also:

getstring
edstring

sethires

Set graphics mode

Usage:

`sethires(` *Mode* `)`

Mode	*0-15*	Alternative screen (some versions)
	+64	Clear the screen on execution

Return value:

Zero

Description:

This function sets the screen into high resolution graphics mode.

One current version, HERCULES, allows two concurrent screens to be maintained. The mode parameter allows selection of these screens.

Mode +64 doesn't clear the screen on execution, thereby allowing you to see the previous image in the video buffer. Note that the CGA doesn't support this feature. Additionally, the HERCULES adapter uses the same video memory for text and graphics. Hence the screen may be corrupted by returning to text mode.

Text mode must be restored, using **settext()**, before leaving a *dGE* module.

Example:

```
sethires(0)
```

will switch to graphics mode, screen 0.

dBASE III:

`&sethires,` *Mode*, `end`

.hires

also:

settext

setpal

Select the colour palette

Usage:

`setpal(` *Bg, Int, Fg* `)`

Bg	Background colour
Int	Foreground intensity (CGA only)
Fg	Foreground colour (CGA only)

Return value:

Zero

Description:

This function sets the colour palette for the CGA colour version and the background colour for the EGA version.

Once palettes have been selected they remain in force until changed again by this function or until a subsequent **sethires**(..) is issued after which they revert to the default values.

CGA

Bg	Colour index in range 0-15	
Int	*0*	normal
	1	high intensity
Fg	*0*	select from green, red, brown
	1	select from cyan, magenta, white

Everything on screen that would normally be black in mono versions will be in the background colour.

Everything on screen that would normally be white in mono versions will be in one of the three foreground colours.

The intensity parameter may have no effect on some monitors.

Example:

```
setpal( 8, 1, 1)
```

will select a backgound of grey, and a foreground in high intensity with a palette consisting of –

1 cyan
2 magenta
3 white

EGA

Only the background colour is affected by this function. The other two parameters are ignored.

Example:

```
setpal( 1, 0, 0)
```

will select a backgound of blue.

dBASE III:

`&setpal,` *Bg, Int, Fg* `end`

settext

Set text mode

Usage:

```
settext( )
```

Return value:

Zero

Description:

This function switches the screen to text mode.

It must be called every time you exit a *dGE* graphics module, prior to showing text on screen.

Example:

```
settext()
```

dBASE III:

```
&settext
```

See also:

sethires

setver

Set mode of operation

Usage:

```
setver( Version )
```

Version *1* *dGE Version 1*
2 *dGE Version 2*

Return value:

Zero

Description:

This function sets the mode of operation of *dGE* to the *dGE Version 1* (compatibility mode) or to *dGE Version 2*. Version 1 functions that have been superseded or re-named are still supported, although not documented here.

The default on installing SETDGE is Version 2. This may be overridden by the /1 switch –

```
SETDGE /1↵
```

The *Version* may be repeatedly switched from within a program if both modes of operation are required.

Version 1 always uses the screen scale 150 x 225. Version 2 uses 1000 x 1350.

Example:

```
setver(1)
```

Sets *dGE Version 1* compatibility mode.

dBASE III:

```
&setver, Version, end
```

setwin

Define statistics window

Usage:

`setwin( X0, Y0, X1, Y1 )`

X0	Lower left X
Y0	Lower left Y
X1	Upper right X
Y1	Upper right Y

Return value:

Zero

Description:

This function defines the window on screen within which statistics functions are drawn.

For compatibility with *dGE Version 1*, if length parameters in the statistics functions are non-zero the origin of the window is taken from the immediately preceding graph functior

In *dGE Version 2* graphs can extend in the negative direction from the origin. The Versio 1 convention would not allow lines to be drawn in negative regions.

In general, therefore, use the **setwin(..)** function when in Version 2 mode.

Example:

```
setwin( 100, 100, 1000, 1000 )
```

will define a rectangular window with bottom left at 100, 100 and top right at 1000, 100

Mean, standard deviation and best-fit lines will be clipped to appear in this window.

dBASE III:

`&setwin, X0, Y0, X1, Y1, end`

setwin

See also:

stats
bestfit
minmax

shad

Fill an area with a patter

Usage:

shade (*X, Y, Pattern, Colour*)

X	X position of an interior point
Y	Y position of an interior point
Pattern	Shade pattern
Colour	0-15

Return value:

Zero

Description:

This function fills the interior of an area bounded by solid lines with a specified fill patte from the *dGE* library.

The point *X,Y* identifies an interior point.

Complex shapes, particularly ones with re-entrant boundaries may not be properly filled this function.

Ideally the shape should be entirely concave.

Recursive methods of filling complex shapes require a great deal of memory. Because of this, in the current version of *dGE* such algorithms have not been used. This may be altered in later versions.

Example:

```
shade( 100, 200, 5, 8 )
```

will fill an area within which 100, 200 is an interior point. The fill pattern is 5 and the colour is 8.

shade

dBASE III:

&shade, *X, Y, Pattern, Colour,* end

See also:

boxfill

stats

Draw statistical lines

Usage:

`stats(` *Wid, Mode, Lstyle, Lcolour*)

Wid	Width of window	
Mode	*0*	Mean only
	1	Mean and standard deviation
Lstyle	*= 0*	Solid
	> 0	Broken with interval of Lstyle pixels
	+128n	Line index (for plotters only)
Lcolour	0-15	

Return value:

Zero

Description:

This function draws lines representing the mean and standard deviation of the data set previously transmitted using **datastore(..)** and shown with one of the graph functions.

The values on which it acts are those of the first parameter in the **datastore(..)**, i.e. *P1*. This has relevance to the high-low-close function.

The standard deviation is shown with a line style twice that of the mean and a colour index +1.

Lines are drawn in a window defined by the positioning of the immediately preceding graph function.

The bottom left of the window is taken from the origin of the graph. The width is given by the Wid parameter.

If Wid is 0 the window is defined by the **setwin(..)** function which must previously have been issued. Use the **setwin(..)** function if your graph extends in the negative X direction

Example:

```
stats( 100, 0, 1, 2 )
```

will draw a mean of the data set with line style 1 and colour 2.

The line will be drawn in a window of width 100 extending in the positive X direction from the origin of the preceding graph.

dBASE III:

`&stats,` *Wid, Mode, Lstyle, Lcolour,* `end`

See also:

setwin
getmean
getsd

timedata

Add a point to a time series graph

Usage:

`timedata(` *Amp1, Amp2, Amp3, Amp4* `)`

Amp1-4 Amplitude of channels 1 to 4
If index channel –
0 no index mark (solid horizontal line)
>0 draw index mark (vertical index height = *Amp*)

Return value:

Zero

Description:

This function adds a further point to a time series graph. See **timegraph(..)** for details of its operation.

Example:

```
timedata( 10, 20, 30, 40)
```

will add a time series data point. Channel 1 amplitude is 10, channel 2 is 20, etc.

dBASE III:

`&timedata,` *Amp1, Amp2, Amp3, Amp4,* `end`

See also:

timegraph

timegraph

Display a time series graph

Usage:

`timegraph` (*X, Y, Xint, Xpts, Ch1, Ch2, Ch3, Ch4*)

`timedata` (*Amp1, Amp2, Amp3, Amp4*)

X	X origin of graph (on right side).	
Y	Y origin	
Xint	X interval between points	
Xpts	Maximum number of points to display	
Ch1-4	*0*	Channel disabled
	1-15	Colour of graph or index line
	+128	Channel is an index
	+256	Superimpose mean
	+512	Superimpose standard deviation
Amp1-4	Amplitude of channels 1 to 4	
	If index channel –	
	0	no index mark (solid horizontal line)
	>0	draw index mark (vertical index height = *Amp*)

Return value:

Zero

Description:

This function displays a time series graph from up to 4 channels of amplitude information.

The graph is in continuous (chained line) format and is drawn at fixed increments in the X axis.

Time increases in the positive X direction.

It differs fundamentally from other data presentation functions in that the function call **timegraph(..)** is issued first before data is "stored". Initially no graph is drawn.

timegraph

As data points are added using the **timedata(..)** function so the graph is built on screen.

The graph is built TO THE LEFT OF THE ORIGIN. The very first point is drawn at the origin.

As further points are added using **timedata(..)** the first point moves to the left and the graph scrolls. The most recent point is always at the origin on the right of the screen, while the least recent point is on the left.

When the number of points exceeds the maximum, *Xpts*, the oldest point, on the extreme left, is discarded.

Up to 4 channels may be shown. Any channel may be disabled.

Any channel may be nominated to be an *index* channel. In this event a continuous line is drawn to the left from the graph origin and vertical index marks are added at points where the amplitude in the time data is non-zero.

The data array is cleared and reset by the **datareset()** function.

Data may be modified by the **datapc(..)** function, but the **datarange(..)** function has no effect.

Example:

```
timegraph( 1200, 675, 10, 100, 5, 6+128, 0, 0)
timedata( 50, 0, 0, 0)
timedata(-50, 10, 0, 0)
```

will begin a time series graph with origin at 1200, 675 and increments of 10 units in the X direction.

Up to 100 data points will be drawn before the oldest is discarded.

Channel 1 is a data channel to be drawn in colour 5.

Channel 2 is an index channel to be drawn in colour 6.

Both other channels are disabled.

The first data point has amplitude 50 and there is no index mark.

The second data point has amplitude -50 and there is an index mark of height 10 units.

BASE III:

`&timegraph, X, Y, Xint, Xpts, Ch1, Ch2, Ch3, Ch4, end`

See also:

labelx
labely
timedata

vecicon

Draw an icon

Usage:

`vecicon(` *Len, Ang, Mode, IconId, Colour* `)`

Len	Length of positional vector (relative)	
Ang	Angle of positional vector	
Mode	*0*	User-defined icon
	1	Library icon
IconId	*mode 0*	Icon code 0 to 9
	mode 1	Icon definition string
Colour	0-15	

Return value:

Zero

Description:

This function prints an icon on the screen. It may be selected from an internal library or defined by the user.

The user-defined icon is a 16 x 16 bit cell composed of the binary pattern of a 32 character string.

The image is composed of 16 double byte rows ordered left to right and in descending row

The least significant bit of each byte is on the left.

Example:

```
vecicon( 100, 45, 1, 5, 10)
```

will draw library icon number 5 at a vector position 100, 45 relative to the current positi in colour 10.

vecicon

dBASE III:

[illegible], [illegible]

Or

&vecicon, *Len, Ang, Mode, Colour,* txt, *IconId,* end

Note the re-ordering of the *Colour* and *IconId* parameters.

See also:

fixpos
movevec

vecstring

Draw a text string

Usage:

`vecstring(` *Len, Ang, Cset, Mode, Colour, String*)

Len	Length of positional vector (relative)	
Ang	Angle of positional vector	
Cset	Character set number 0 or 1	
Mode	*0*	draw horizontally
	1	draw vertically
	+2	don't clear (i.e. superimpose)
	+8	position by centre
	+16	position by right corner
Colour	0-15	
String	Text string	

Return value:

Zero

Description:

This function draws a text string at a vector position relative to the current position. It may be drawn either horizontally or vertically (with descending characters), using one of the two current character sets.

Optionally the line is not cleared before superimposing the characters.

It may be left (default), mid or right justified.

Example:

```
vecstring( 100, 45, 0, 0, 7, "String..")
```

will draw the string `String..` at vector offset 100, 45 from the current position, using character set 0 and colour 7.

vecstring

dBASE III:

```
&vecstring, Len, Ang, Cset, Mode, Colour, txt, String, end
```

See also:

fixpos
movevec

xyaxes

Draw a set of X-Y axes

Usage:

xyaxes(*X, Y, Xlen, Ylen, Xdivs, Ydivs, Mode, Colour*)

X	X origin
Y	Y origin
Xlen	Length of arm of axis in X direction
Ylen	Length of arm of axis in Y direction
Xdivs	Divisions per arm in X direction
Ydivs	Divisions per arm in Y direction
Mode	Defined by 8 bit number

Bit	Meaning if set	Decimal
0	Draw negative Y arm	+1
1	Draw negative X arm	+2
2,3	Draw Y grids .. variable pitch	+4
4,5	Draw X grids .. variable pitch	+16
6	Draw box frame round exterior	+64

Colour Colour of axes and grids, 0-15

Return value:

Zero

Description:

This function draws a set of X-Y axes with scale marks at regular intervals. The axes are positioned on screen by X,Y. They may include negative extensions in the X or Y directions.

Optionally grids may be superimposed in either X or Y, and the entire axes may be framed by a box.

The line style of the grids is dictated by the two bit pattern as follows –

2nd bit	1st Bit	Line style
0	1	Continuous
1	0	Broken line with pitch 2
1	1	Broken line with pitch 4

If grid lines are drawn ticks are omitted. Note that by calling this function repeatedly with different parameters complex axes with major and minor grids can be created.

Example:

```
xyaxes(100, 200, 300, 400, 10, 20, 1+4*2+16*3+64, 3)
```

will draw axes at 100, 200 with X length 300, Y length 400 and with 10 divisions in X, 20 divisions in Y.

Y grids are pitch 2, X grids are pitch 4.

dBASE III:

`&xyaxes,` *X, Y, Xlen, Ylen, Xdivs, Ydivs, Mode, Colour,* `end`

See also:

labelx
labely

xygraph

Draw an X-Y graph

Usage:

`datastore(` *Yamp, Icon, Xpos, 0* `)`

`xygraph(` *X, Y, Xinc, Mode, Colour* `)`

Yamp	Amplitude in Y	
Icon	Icon code	
Xpos	X position relative to origin in *scatter* mode	
X	X origin of graph	
Y	Y origin of graph	
Xinc	*0*	Portray in *scatter* format
	> 0	Increment in X
Mode	*0*	Chained lines
	1	Symbols
	2	Chained lines + symbols
	3	Vertical sticks
Colour	Colour of lines and symbols, 0-15	

Return value:

Zero

Description:

This function draws a graph in X-Y (cartesian) format.

If *Xinc* is 0 the graph is drawn in scatter format using X values stored with the **datastore(..)** function.

Example:

```
datastore( 200, 1, 100, 0 )
datastore( 300, 1, 200, 0 )
datastore( 400, 1, 300, 0 )
xygraph( 100, 100, 0, 0, 8 )
```

will draw an X-Y graph using symbols only with origin at 100, 100.

It is drawn in scatter format with X positions 100, 200, 300 and amplitudes 200, 300, 400.

dBASE III:

`&xygraph` *X, Y, Xinc, Mode, Colour,* `end`

See also:

labelx
labely
xyaxes