/***

PGL
Portable Graphics Layer

Documentation

Ver. 1.0

1/11/91

***/

TABLE OF CONTENTS

DETAILED TABLE OF CONTENTS

Portable Graphics Layer
Software Farm
www.swfm.com

Thank you for purchasing the finest portable graphics interface available. We will do our best to give you many years of enjoyment and satisfaction.

The Portable Graphics Layer (PGL) provides a high performance, broad, multi-level interface to various standard graphics/windowing systems. This therefore aids application development by allowing the software engineer to devote time and effort on writing the application and not on the graphics routines that may be required. Also, by recompiling, the application will run

on any one of the hardware/software platforms supported now and in the future by PGL. Finally, the application will not have to cripple its graphics interface on any platform because PGL provides a consistent base of functionality on all platforms, regardless of the underlying deficiencies.

PGL provides both an immediate and a retained (displaylist) graphics interface.

The hardware platforms currently supported include:

IBM PC/XT/AT clones with DOS and a CGA, EGA, VGA and VEGA enhanced EGA graphics cards.
SUN 3/XX, 4/XX, 386i, and Sparcstation Series workstations under SunOS 3.x and 4.x.

The windowing systems currently supported include:

PGL Windows (built in windowing support for systems without such support such as stock PC's).
SunView.
X Window System, Version 11 release 3.

Platforms and windowing systems that are soon to be supported include:

MicroSoft Windows.
Hercules Graphics cards in IBM PC/XT/AT clones.
Unix SystemV/386 support for abovementioned graphics cards.
View2/OpenLook
Motif

Graphics metafile standards to be supported include:

CGM
PostScript

The level of functionality of all aspects of PGL is steadily increasing. The actual direction and content of any new feature is often motivated by requests and suggestions of PGL users and other interested parties. If there is a feature you desire that is either not mentioned here, or is currently unimplemented, please communicate your needs and we will try our best to get it into PGL as quickly as possible.

Rendering Functionality

Currently PGL is capable of rendering lines, polylines, ellipses, arcs, bitmap/stroke font text, filled(pattern/solid)/raster-oped/unfilled rectangles,
and polygons.
Two to 256 colors are supported with inherent capability for approximately 4 billion colors. Write modes supported are 'replace' and 'xor' at this time.

Windowing Functionality

Currently PGL supports the concept of a world coordinate space which is mapped to a viewport in a window on a virtual screen. The world space is comprised of all 32bit integers, the viewport and virtual screen of 16bit integers. The virtual screen effectively isolates the application from the real (variable) size of the screen (either whole display device or system window on the display device).

DisplayList Functionality

There is support at this time for six types of graphical 'objects': picture, device, window, view, segment, and primitive. Primitives are created and collected into segments, which can be nested in other segments.
Devices, windows and views comprise a picture object which is what is displayed.
It is possible to draw, insert, copy, delete, traverse (forwards and backwards)
and perform other operations on any object. Support is also provided for picking and searching 'through' objects.

There is also support for specification and manipulation of a attribute object or 'graphical context'.

Event Functionality

Events are generated by the mouse, keyboard, and by the application. Specific types of events may be ignored or requested for each view, and a procedure to be called when these events occur may be registered for each view. Mouse clicks, double clicks and cursor locking (no automatic tracking) are supported. Cursors images may be redefined and/or hidden.

High Level Interaction Functionality

There is support in PGL for the application to easily zoom, pan, and move objects. Application boundary conditions are specified and the rest is handled by the appropriate PGL functions.

Software written with PGL must include the file

```
#include "graphics.h"
```

This file includes all the include files that are needed by the application now and in the future. The sub-include files and their descriptions
follow:

gtype.h	The typedefs for all types used by PGL are here.
gappl.h	This is the main application include file. It contains information for most of the common PGL functions.
gkbm.h	This file contains the definitions of the

keyboard/mouse interface.

garray.h This file contains all the garray structure definitions for the Ggetarray, Gexecarray and Gapplyarray functions.

gerror.h This file contains the list of error codes.

garg.h This file contains the list of vararg keywords.

gdebug.h This file contains structure definitions for the debug interface.

glowface.h This file contains information about the low level hardware interface.

gproto.h This file contains all the functional prototypes for all the public graphics functions.

Graphics Data and Object Types

Data types are used to indicate the size of a particular value. It is very important to specify the type of a value passed as a parameter in a function call when the application is to be run on machines that have differing
ideas of what the size of an integer (type int) is (i.e. the PC considers the type int to be 16 bits and the SUN considers it to be 32 bits long).

Coordinate types are:Used for:

GT_COORDWorld coordinate type: for primitives, view world dimensions

GT_VCOORDViewport coordinate type: for view viewport dimensions.

GT_SCOORDScreen coordinate type: for window dimensions.

GT_DCOORDDevice coordinate type: for very low level draw routines.

also

GT_UCOORDUnsigned world coordinate type: for unsigned quantities such as height, width, etc.

GT_UVCOORDUnsigned viewport coordinate type.

GT_USCOORDUnsigned screen coordinate type.

Event types are:

GT_EVENTThe type of the event packet returned by the

event routines.

GT_EVENTTYPE Events returned have an ID of this type indicating the general type of event.

GT_EVENTVALUE The specific event is represented by a value of this type (i.e. the ASCII representation or a key).

GT_TIME Events have a time stamp of this type.

Graphics object handles have types:

GT_OBJECT General graphical object type. All such objects may be manipulated using the general object functions.

Types associated with the graphic context:

GT_CTX The type of the graphical context handle.

GT_CTXTYPE The type of the type of graphical context.

GT_ATT All attributes in a graphical context are of this type. This means all colors, pattern types, line widths, are of this type.

GT_UATT The type of all user attributes in a context.

General types:

GT_BITMAP The type of the graphics bitmap handle.

GT_WSTYLE The type of the graphics window style handle.

GT_DRIVER The type of the hardware device driver structure.

GT_EXTREMA The type of the dimensional size structure.

GT_TYPE Object identification type: every graphics object has an ID of this type.

GT_SCALE Multiplicative scale factors are specified by a value of this type: (Segment call's have a scale parameter of this type).

GT_ANGLE Rotation angles are of this type: (Segment calls have a angle parameter of this type).

GT_TAG All graphics objects have tags that are of this type.

GT_VIDEOMODE Videomodes are specified by a value of this

type.

Function pointer types are

GT_FNPTRGeneral pointer to a function which returns int.

GT_VOIDFNPTRGeneral pointer to function which returns void.

and general portable types are:

GT_INT8 A type which is 8 bits long.
GT_UINT8

GT_INT16A type which is 16 bits long.
GT_UINT16

GT_INT32A type which is 32 bits long.
GT_UINT32

Naming Scheme

There is an attempt to keep graphics names logical and at least identifiable. In that vein all names have a prefix of some sort.

Public:

*Functions:

General graphics functions

G Capital letter 'G' and lowercase letters following.

Graphics low level rendering function pointers

Gr_ Capital letter 'G', lowercase 'r', underscore, with lowercase letters following.

*Data:

There is no public graphics data.

*Values:

General graphics values (defines)

G_ Capital letter 'G', underscore, with uppercase letters following.

Graphics object types

GO_ Capital letters 'GO', underscore, with uppercase letters

following.

Graphics keywords (used largely as variable argument function parameters.

GARG_ Capital letters 'GARG', underscore, with uppercase letters following.

*Types:

General typedefs used by graphics and applications

GT_ Capital letters 'GT', underscore with uppercase letters following.

Internal:

*Functions:

General internal functions

g_ Lowercase g, followed by an underscore and lowercase letters.

*Data:

gd_ Lowercase g, lowercase d, followed by an underscore and lowercase letters.

*Values:

GI_ Uppercase letter 'G', 'I', followed by an underscore and uppercase letters.

PGL SYSTEM MANAGEMENT

Ginit() initialize graphics systems.

Gtermin() terminates the graphics system.

Gconfig(...) configure the graphics system.

Ginqconfig(...) inquire the configuration of the graphics system.

Ginit PGL SYSTEM MANAGEMENT Ginit

DEFINITION

Initializes the graphics system. This includes:

1. Creating the default window style unless the application has created and open one already. This window style is applied to the root window on systems where PGL WINDOWS is the native window system.

PURPOSE

To tell the graphics system to initialize itself.

OPERATION

int Ginit()

Returns zero if successful.

EXAMPLES

BUGS

SEE ALSO

Gwstyle, Gopenwstyle, Gdevice, Gtermin, Gconfig.

DEFINITION

Terminates the graphics system. This includes:

1. Closing all devices. This sends a termination message to the window system and hardware drivers.

2. Freeing memory associated with the internal graphics system.

PURPOSE

To tell the graphics system to terminate itself.

OPERATION

void Gtermin()

EXAMPLES

BUGS

SEE ALSO

Gwstyle, Gopenwstyle, Gdevice, Ginit, Gconfig.

DEFINITION

Configures the graphics system.

PURPOSE

To set and change internal elements of the graphics system, both before and after initialization, that can not be changed by any other methods.

OPERATION

void Gconfig(...)

This function takes GARG_ keyword, value pairs as parameters.

Valid keywords are:

GARG_NUMPOLYGONVLINESint Maximum number of edges a horizontal slice though a filled object will encounter. This will allocate memory to support the fill algorithm for this worst case polygon. G_DEFAULT_NUMPOLYVLINES is the default.

*Inquire

void Ginqconfig(...)

This function inquires the current configuration with GARG keyword, value address pairs. All keywords described above are valid.

EXAMPLES

BUGS

SEE ALSO

Gwstyle, Gopenwstyle, Gdevice, Ginit, Gtermin.

PGL OBJECTS

There are six groups of objects: primitives, segments, views, windows, devices, and pictures. Primitives are discussed in detail in another section. Segments are ordered lists of primitives and other segments and are therefore essentially hierarchical structures of graphic data. Views specify how and where

the data is to be rendered. Windows are user manipulated areas on a video device
providing the 'paper on a desktop' metaphor. Devices are a representation of the output hardware and it's characteristics. Pictures are symbolic description
of the pipeline which takes graphic data to the graphical output device.

The graphics segment object is used to arrange the graphic data primitives(which consists of lines, ellipses, text, etc.) into hierarchical structures. The structure (which is completely determined by the application) is composed of segments within segments and the use of the call primitive. The call primitive treats the segment like a procedure by 'calling' the segment
and then immediately returns to the primitive following the call. Therefore a segment A may call other segments B ands C many times, and segments B and C may be called by many other segments or may call segments themselves.

For example, a segment may contain the graphics describing a 'confirm or cancel' button. There may be many menus displayed, each of which display this button. The menus would therefore each have a call to the one segment containing the button's description. (Note that the call primitive has an optional translation factor built in).

The graphics view object is used to specify both what objects are to be drawn (determined by the size of the world boundary) and to what area on the screen they will be drawn to (determined by the size of the viewport boundary). Note that the word 'boundary' is meant to specify a rectangular area which is a subset of a larger rectangular area of the same type.

The world boundary is a rectangular area which is a subset of the entire world coordinate space. The size of the entire world coordinate space is determined by the nature of the coordinates it uses which are the set of 32 bit integers at this time.
I.E. the world boundary is a subset of the rectangular bounds:

{(-2147483648, -2147483648), (+2147483647, +2147483647)}.

Boundaries are specified by the coordinates of their lower left hand and upper right hand corners. The world coordinate space is the area and coordinates
that all graphics object primitives (such as lines and circles) are specified in.

The viewport boundary is a rectangular area which is a subset of the entire viewport coordinate space. Whenever a window is associated with a view, the window maps to a subset of the viewport coordinate space, namely

{(-32768, -32768), (+32767, +32767)}.

For example, suppose one wants to draw all primitives located between (0,0) and (1000,1000) to the upper half of a window on the monitor. Then one would specify a world boundary of (0,0),(1000,1000) and a viewport boundary of (-32768, 0),(+32767, +32767).

The graphics window object is used by the application to customize and manipulate the windows on video devices. These windows may be system level

windows run by the likes of SunView or TWM under the X Window system. Or they may be run by the window manager internal to PGL(called PGL Windows). Any sub-window may be a PGL window and all windows on stock PCs are PGL windows. Subwindows are regular windows except that they are confined to the area of their parent window.

The window boundary is a rectangular area which is a subset of the entire screen coordinate space. Whenever a device is associated with a window, the device maps to a subset of the window coordinate space, namely

{(-32768, -32768), (+32767, +32767)}.

For example, suppose one wants to position a window in the upper left corner of the video screen. Then one would specify a screen boundary of (-32768,0),(0,+32767).

The graphic device object is used to reference and configure the output hardware(i.e. a video screen). Video, mouse and keyboard drivers are installed in the device object and when the device is opened, these drivers are initialized. Subsequently the device may be inquired as to the size of the device (in device coordinates), maximum number of colors of the device, etc. The drivers may be changed effecting an immediate termination of the old driver and initialization of the new. In this way video modes may be changed (i.e. in this way one can change PC's EGA mode to VGA mode at run time).

The graphic picture object is used to specify the display pipeline, which consists of the graphics data to draw, the view to use to transform the graphics data, the window to display in, and the device to output to. The picture is designed to allow maximum flexibility when mixing and matching the objects in the display pipeline.

SEGMENT PGL OBJECTS
SEGMENT

DEFINITION

A segment is ordered list of primitives and other segments. When a segment is created there are no required parameters and it is bereft of any data. If it is the open segment (opened by a invoking 'Gopen' or 'Gpushopen') then subsequent primitives are put into the segment in sequential order.

TYPE

GO_SEGMENT
GO_CSEGMENT

ATTRIBUTES

GARG_HIDDEN

Specifies whether the segment is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Gsegment(va_alist)
va_dcl

GT_OBJECT Gcsegment(va_alist)
va_dcl


Allowable keywords:

      GARG_NAME(char *)segment_name
                  default "generic seg"

      GARG_PARENTGT_OBJECT parentseg
                  G_OPENOBJECT(i.e. open segment)
                  NULL        (default)

      GARG_COMPILEDTRUE
                  FALSE       (default)

      GARG_ANGLE(GT_ANGLE )rotation in degrees
                  default is 0

      GARG_TRANSX(GT_COORD )x translation to add to all
                  data in the segment.
                  default is 0.

      GARG_TRANSY(GT_COORD )y translation to add to all
                  data in the segment.
                  default is 0.

Gcsegment() creates a compiled segment graphics object. It is
the same as Gsegment() except that the default for
GARG_COMPILED is TRUE.


GT_OBJECT Gexecsegment(values)
struct garraysegment *values;

Takes as input an address of a garraysegment structure and the
creates the corresponding segment graphics object.
```

*Destroy

```
void Gdelete(segment);
GT_OBJECT segment;
```

*Inquire

```
void Ginqsegment(segment, va_alist)
GT_OBJECT segment;
```

va_dcl

Takes the input segment graphics object and returns the values requested. Valid 'GARG_' keywords are the same as those for segment create.

*Modify

```
void Gmodsegment(segment, va_alist)
GT_OBJECT segment;
va_dcl
```

Takes the input segment graphics object and modifies the values requested. Valid 'GARG_' keywords are the same as those for segment create.

*Standard Object Operations

All operations are valid. Refer to the particular operation for descriptions in the 'Standard Object Operations' section.

*Draw

```
Gdraw(segment)
GT_OBJECT segment;
```

Renders the graphics object using the current open picture to determine the world to device transform, the clipping bounds, and the window and device to draw to.

CHILD OBJECTS

Primitives, segments.

PARENT OBJECTS

Segments
Calls
Views

EXAMPLES

SEE ALSO

Primitives, Segments, Calls,

DEFINITION

A view is a graphics object that contains information that describe the viewing parameters of a displayable object. It contains two user

specified rectangles, one in world and one in viewport coordinate space. This provides the mapping from an area in world space (where graphics primitives exist) to a viewport (an area in a window). A graphics data segment is associated with the view to indicate what data exists in the view. A picture is created to actualize the view and make it visible in the specified window on the specified device.

TYPE

GO_VIEW

ATTRIBUTES

GARG_HIDDEN

Specifies whether the view is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Gview(va_alist)
va_dcl
```

Allowable keywords:

```
GARG_NAME(char *)view_name
            default "generic view"

GARG_PARENTGT_OBJECT parentseg
            G_OPENOBJECT(i.e. open segment)
            NULL        (default)

GARG_SEGMENT(GT_OBJECT )segment;
            NULL        (default)

GARG_VIEWPORT(GT_VCOORD )vxmin,
            (GT_VCOORD )vymin,
            (GT_VCOORD )vxmax,
            (GT_VCOORD )vymax,

GARG_WORLD(GT_COORD )wxmin,
            (GT_COORD )wymin,
            (GT_COORD )wxmax,
            (GT_COORD )wymax,

GARG_PROC(GT_FNPTR )eventproc
            NULL              (default)

GARG_EVENTMASK(GT_EVENTMASK )eventmask
            G_DEFAULTEVENTMASK(default)

GARG_CONSUMEMASK (GT_EVENTMASK )consumemask
```

```
      GT_OBJECT Gexecview(values)
      struct garrayview *values;

      Takes as input an address of a garrayview structure and the
      creates the corresponding view graphics object.

*Destroy

      void Gdelete(view);
      GT_OBJECT view;

*Inquire

      void Ginqview(view, va_alist)
      GT_OBJECT view;
      va_dcl

      Takes the input view graphics object and returns the values
      requested. Valid 'GARG_' keywords are the same as those for
      view create.

*Modify

      void Gmodview(view, va_alist)
      GT_OBJECT view;
      va_dcl

      Takes the input view graphics object and modifies the values
      requested. Valid 'GARG_' keywords are the same as those for
      view create.

*Standard Object Operations

      All operations are valid. Refer to the particular operation for
      descriptions in the 'Standard Object Operations' section.

*Draw

      Gdraw(view)
      GT_OBJECT view;

      Renders the graphics object using all the pictures that have
      been associated with it, each determining the world to device
      transform, the clipping bounds, and the window and device to
      draw to.

CHILD OBJECTS

      None.

PARENT OBJECTS
```

None.

EXAMPLES

SEE ALSO

Segments, Pictures

WINDOW PGL OBJECTS WINDOW

DEFINITION

A window is a graphics object that represents an area on the video display. This area is usually created and managed by the resident window system/window manager. Which window system/manager is used is determined by the device associated with the pictures that include the window. It may or may not be bordered, movable, resizable, iconifyable, etc. These parameters are determined by the windows 'window manager style' (of type GT_WSTYLE).

TYPE

GO_WINDOW

ATTRIBUTES

GARG_HIDDEN

Specifies whether the window is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Gwindow(va_alist)
va_dcl
```

Allowable keywords:

```
GARG_NAME(char *)window_name
            "generic window name" (default )

GARG_SCREEN(GT_SCOORD )sxmin
            (GT_SCOORD )symin
            (GT_SCOORD )sxmax
            (GT_SCOORD )symax

GARG_MAXNUMCOLORS (GT_ATT )numcolors
            G_DEFAULT_MAXNUMCOLORS(default)
```

```
                GARG_WINDOWMANAGERSTYLE (GT_WSTYLE )wstyle
                            open window manager style (default)

                GARG_PARENTGT_OBJECT parentwindow
                            G_OPENOBJECT(i.e. open segment)
                            NULL        (default)

          GT_OBJECT Gexecwindow(values)
          struct garraywindow *values;

          Takes as input an address of a garraywindow structure and the
          creates the corresponding window graphics object.

     *Destroy

          void Gdelete(window);
          GT_OBJECT window;

     *Inquire

          void Ginqwindow(window, va_alist)
          GT_OBJECT window;
          va_dcl

          Takes the input window graphics object and returns the values
          requested. Valid 'GARG_' keywords are the same as those for
          window create.

     *Modify

          void Gmodwindow(window, va_alist)
          GT_OBJECT window;
          va_dcl

          Takes the input window graphics object and modifies the values
          requested. Valid 'GARG_' keywords are the same as those for
          window create.

     *Standard Object Operations

          All operations are valid. Refer to the particular operation for
          descriptions in the 'Standard Object Operations' section.

     *Draw

          Gdraw(window)
          GT_OBJECT window;

          Renders the graphics object using all the pictures that have
          been associated with it, each determining the world to device
          transform, the clipping bounds, and the data and device to
          draw to.

CHILD OBJECTS
```

Possible subwindows.

PARENT OBJECTS

A window if a subwindow.

EXAMPLES

SEE ALSO

Segments, Pictures, Wstyle

DEFINITION

A device is a graphics object that represents the interface to a window and hardware system configuration.

TYPE

GO_DEVICE

ATTRIBUTES

There are no attributes used at this time.

OPERATIONS

*Create

```
GT_OBJECT Gdevice(va_alist)
va_dcl
```

Allowable keywords:The possible parameter values:

GARG_VIDEODRIVERTYPE	int	system video modes user defined video type G_DEFAULT_VIDEOMODE (default) When the videomode changes there will be the visible effect consisting of the clearing of the window(s) and resizing (where necessary) and repainting of same.
GARG_VIDEODRIVER	GT_DRIVER	user_defined_video_driver (defaults are supplied by system when a system video hardware type is chosen

```
                        with the GARG_VIDEODRIVERTYPE option).

GARG_MOUSEDRIVERTYPEintsystem mouse type
                        G_DEFAULT_MOUSE(default)

GARG_MOUSEDRIVER GT_DRIVER user_defined_mouse_driver
                        (default is supplied by system when a
                        system mouse hardware type is chosen
                        with the GARG_MOUSEDRIVERTYPE option).

GARG_KEYBOARDDRIVERTYPE intsystem keyboard type
                        G_DEFAULT_KEYBOARD(default)

GARG_KEYBOARDDRIVERGT_DRIVER user_defined_keyboard_driver
                        (default is supplied by system when a
                        system keyboard hardware type is chosen
                        with the GARG_KEYBOARDDRIVERTYPE option).

GARG_MOUSEENABLED intG_TRUE(default) or G_FALSE

GARG_VIDEOENABLED intG_TRUE(default) or G_FALSE

GARG_KEYBOARDENABLEDintG_TRUE(default) or G_FALSE

GARG_MOUSEBUTTONSENABLED intmask of valid mouse buttons
                        G_DEFAULT_MOUSEBUTTONSENABLED  (default)

The following window manager and driver options are applied to
all windows SUBSEQUENTLY CREATED whenever picture graphic objects
are created on this, the given device.

GARG_WINDOWTYPEinttype of window driver system.
                        G_NATIVEWINDOWTYPE(default)
                        other system supplied options are:

                              G_PGLWINDOWTYPE

                        see the file "gappl.h". If the
                        windowtype is not a system supplied
                        type then the given value is stored
                        but otherwise ignored. Note: PGLWINDOWS
                        is a window system built in to PGL and
                        is the native window type on stock PCs
                        and can be a subwindowtype on any other
                        window system.

GARG_WINDOWDRIVER struct gwindowdriver * user_window_driver
                        (defaults are supplied by system when a
                        system window type is chosen with the
                        GARG_WINDOWTYPE option).

GARG_WINDOWMANAGERTYPEinttype of high level PGL window manager.
                        G_NATIVEWINDOWMANAGER(default)
```

```
GARG_WINDOWMANAGERstruct gwsystem *wmanager

Sub windows can be different than their parent window. The following
apply to any SUBWINDOWS subsequently created during an instantiation
of any picture graphic objects on this given device.

GARG_SUBWINDOWTYPEinttype of window driver system.
                        G_NATIVEWINDOWTYPE(default)
                        other system supplied options are:

                              G_PGLWINDOWTYPE

                        see the file "gappl.h". If the
                        windowtype is not a system supplied
                        type then the given value is stored
                        but otherwise ignored.

GARG_SUBWINDOWDRIVER struct gwindowdriver * user_window_driver
                        (defaults are supplied by system when a
                        system window type is chosen with the
                        GARG_WINDOWTYPE option).

GARG_SUBWINDOWMANAGERTYPEinttype of high level PGL window
                              manager.
                        G_NATIVEWINDOWMANAGER(default)

GARG_SUBWINDOWMANAGERstruct gwsystem *wmanager

GARG_NAME(char *)device name
            "generic window name" (default )


      GT_OBJECT Gexecdevice(values)
      struct garraydevice *values;

      Takes as input an address of a garraydevice structure and the
      creates the corresponding device graphics object.

*Destroy

      void Gdelete(device);
      GT_OBJECT device;

*Inquire

      void Ginqdevice(device, va_alist)
      GT_OBJECT device;
      va_dcl

      Takes the input device graphics object and returns the values
      requested. Valid 'GARG_' keywords are the same as those for
      device create.

*Modify
```

```
            void Gmoddevice(device, va_alist)
            GT_OBJECT device;
            va_dcl

            Takes the input device graphics object and modifies the values
            requested. Valid 'GARG_' keywords are the same as those for
            device create.

      *Standard Object Operations

            All operations are valid. Refer to the particular operation for
            descriptions in the 'Standard Object Operations' section.

      *Draw

            *** NOT IMPLEMENTED ***

            Gdraw(device)
            GT_OBJECT device;

            Renders the graphics object using all the pictures that have
            been associated with it, each determining the world to device
            transform, the clipping bounds, and the data and device to
            draw to.

CHILD OBJECTS

      None.

PARENT OBJECTS

      None.

EXAMPLES

BUGS

      Very high level windowmanagers are not supported (i.e. the work is
      all done internal to PGL and it won't take advantage of the power of
      some of the modern window systems today).


SEE ALSO

      Segments, Pictures, Drivers(low level interface).


PICTURE                          PGL OBJECTS
PICTURE

DEFINITION

      A picture is the link between the video device and the graphics data the
```

application has created. In a real sense only picture graphic objects are seen by the user. This link consists of a segment graphic object (with graphic primitive objects in it representing lines, circles, etc.), a view graphic object(representing how the graphics data will look), a window graphic object(representing the area on the screen the data is drawn to as well as the system level window to draw into), and a device graphic object(representing the actual hardware used for output which may be video, RAM, plotters, etc.).

PURPOSE

The picture provides methods to control the conceptual 'viewing pipeline', the link between a graphical action by the application and the resultant graphical output. Thus multiple views(one set of data viewed differently at different areas of the window), multiple windows of the same view, and multiple devices for output.

TYPE

GO_PICTURE
GO_SYMBOLICPICTURE

ATTRIBUTES

GARG_HIDDEN

Specifies whether the picture is visible or not.

GARG_FILL

Specifies if and how the picture background is filled (if at all).

OPERATIONS

*Create

```
GT_OBJECT Gpicture(view, window, device)
GT_OBJECT view;
GT_OBJECT window;
GT_OBJECT device;
```

A picture graphic object is created and it's handle is returned. If the system managed window associated with the window graphic object does not yet exist, it is created. If there is not an open device, the default device is opened.

If G_WILDCARD is a parameter in place of any of the given objects, then a symbolic picture graphic object is created. This symbolic picture represents all pictures that exist, whenever the symbolic picture is referenced, that have a view, window, and device which match the symbolic picture's view, window, and device.
For example, if all three parameters are G_WILDCARD, then the

symbolic picture represents all pictures created. Therefore at initialization, drawing the symbolic picture draws nothing, and at any other time, it will draw everything created.

view

If view is equal to: 1. a 'view' graphic object

Then the given view will be linked with the given window and given device.

2. G_OPENOBJECT

Then the currently open view will be linked with the given window and given device.

3. (GT_OBJECT)G_WILDCARD

Then all view graphic objects will be linked with the given window and given device.

window

If window is equal to: 1. a 'window' graphic object

Then the given window will be linked with the given view and given device.

2. G_OPENOBJECT

Then the currently open window will be linked with the given view and given device.

3. (GT_OBJECT)G_WILDCARD

Then all window graphic objects will be linked with the given view and given device.

device

If device is equal to: 1. a 'device' graphic object

Then the given device will be linked with the given window and given view.

2. G_OPENOBJECT

Then the currently open device will be linked with the given window and given view. If there is no open device, the default device will be created and opened automatically.

3. (GT_OBJECT)G_WILDCARD

Then all device graphic objects will be linked with the given window and given view.

```
GT_OBJECT Gexecpicture(values)
struct garraypicture *values;
```

Takes as input an address of a garraypicture structure and the creates the corresponding picture graphics object.

*Destroy

```
void Gdelete(picture);
GT_OBJECT picture;
```

*Inquire

```
void Ginqpicture(picture, va_alist)
GT_OBJECT picture;
va_dcl
```

Takes the input segment graphics object and returns the values requested. Valid 'GARG_' keywords are:

GARG_VIEW

GARG_WINDOW

GARG_DEVICE

*Modify

There is no way, and no sense, to modifying a picture object.

*Standard Object Operations

All operations are valid. Refer to the particular operation for descriptions in the 'Standard Object Operations' section.

*Draw

```
Gdraw(picture)
GT_OBJECT picture;
```

Renders the graphics object using the picture to determine the world to device transform, the clipping bounds, and the window and device to draw to, and view graphic data to draw.

*Traversal

```
GT_OBJECT Gfirstpicture(obj)
GT_OBJECT obj;
```

Returns the first picture in an internal list associated with the given object. NULL is returned if there is no such picture. So that if the graphic object 'obj' is a:

VIEW — The picture which is drawn first when the view is drawn is returned.

WINDOW — The picture which is drawn first when the window is drawn is returned.

DEVICE — The picture which is drawn first when the device is drawn is returned.

```
GT_OBJECT Glastpicture(obj)
GT_OBJECT obj;
```

Returns the last picture in an internal list associated with the given object. Similar to Gfirstpicture, this is the picture drawn last when the given object 'obj' is drawn. NULL is returned if there is no such picture.

```
GT_OBJECT Gnextpicture(obj, picture)
GT_OBJECT obj;
GT_OBJECT picture;
```

Returns the next picture following the given picture in an internal list associated with the given object 'obj'. NULL is returned if there is no such picture.

```
GT_OBJECT Gprevpicture(obj, picture)
GT_OBJECT obj;
GT_OBJECT picture;
```

Returns the previous picture preceding the given picture in an internal list associated with the given object 'obj'. NULL is returned if there is no such picture.

```
int Gexcisepicturefromobject(picture, object)
GT_OBJECT picture;
GT_OBJECT object;
```

Removes the given picture from the list of pictures associated with the given object. The object must be the view, window or device object which is a component of the given picture. The next time the given object is drawn, the given picture will not be drawn. Non-zero is returned if the given picture is not associated with the given object.

```
int Gappendpicturetoobject(picture, object)
GT_OBJECT picture;
GT_OBJECT object;
```

Appends the given picture to the end of the list of pictures associated with the given object. The object must be the view, window or device object which is a component of the given picture. The next time the given object is drawn, the given picture will be drawn last. Note that a Gexcisepicturefromobject combined with a Gappendpicturetoobject is effectively a picture 'pop-up'. Non-zero is returned if the object is of an incorrect type.

```
int Gprependpicturetoobject(picture, object)
GT_OBJECT picture;
GT_OBJECT object;
```

Prepends the given picture to the beginning of the list of pictures associated with the given object. The object must be the view, window or device object which is a component of the given picture. The next time the given object is drawn, the given picture will be drawn first. Note that a Gexcisepicturefromobject combined with a Gprependpicturetoobject is effectively a picture 'pushunder'. Non-zero is returned if the object is of an incorrect type.

```
void Gpopup(picture)
GT_OBJECT picture;
```

'Pop-up' the system window associated with the given picture object. This makes the window the top window and completely unobscured.

```
void Gpushunder(picture)
GT_OBJECT picture;
```

'Pushunder' the system window associated with the given picture object. This makes the window the bottom window and possibly overlapped by other windows.

CHILD OBJECTS

None.

PARENT OBJECTS

None.

EXAMPLES

SEE ALSO

View, Window, Device.

PRIMITIVES

A primitive is a graphics object that cannot contain any graphics objects within it. It is a leaf of the graphics data structure and usually represents a visual entity.

I. Creation

When primitive graphics objects are created, they are put in the currently open segment inheriting the attributes found in the currently open context. A handle of the created object is returned to the caller. The primitive object is not checked to determine wether it (and with the attributes in it's graphical context) will make the primitive exceed the bounds of world space. Coordinate points are always specified in window world coordinates and are of type GT_COORD.

Compiled primitive graphics objects are slightly different in that they allocate a little more memory to have space to save a copy of the device coordinates of the primitive at the last screen/viewport/world transformation model it was last drawn in. This saves time when it is drawn again by the CPU not having to spend the time to recalculate the transformations.

primitives usually therefore usually exist inside 'segments' which are provided to allow various tree structures to be maintained.

```
GT_OBJECT Garc(xc, yc, a, b, start, end)
GT_OBJECT Gcarc(xc, yc, a, b, start, end)
GT_OBJECT Gcall(seg, tx, ty, angle, scale)
GT_OBJECT Gccall(seg, tx, ty, angle, scale)
GT_OBJECT Gellipse(xc, yc, a, b)
GT_OBJECT Gcellipse(xc, yc, a, b)
GT_OBJECT Gline(x1, y1, x2, y2)
GT_OBJECT Gcline(x1, y1, x2, y2)
GT_OBJECT Gpline(array, number)
GT_OBJECT Gcpline(array, number)
GT_OBJECT Gpolygon(linelist, arclist)
GT_OBJECT Gcpolygon(linelist, arclist)
GT_OBJECT Grect(x1, y1, x2, y2)
GT_OBJECT Gcrect(x1, y1, x2, y2)
GT_OBJECT Gtext(x, y, string)
GT_OBJECT Gctext(x, y, string)
```

II. Inquire

There are functions provided to provide information about a graphics object to an application. These are called inquire functions. Some inquire the parameters of any type of object and some inquire only about their specific type of object. Some general inquire functions such as Ggetarray and Ggetextrema are discussed elsewhere. Specific inquire functions, one for each graphical object, are the following: (note: replace [objectname] with an objects generic name like 'line' or 'rectangle', etc.)

```
Ginq[objectname](obj, [addresses of]parameter list as it appears
                  for create of the object)
```

```
GT_OBJECT obj;
[pointers to] parameter list as it appears for create of the object;

i.e. for line the function is defined as:

            void Ginqline(obj, x1, y1, x2, y2)
            GT_OBJECT obj;
            GT_COORD *x1, *y1, *x2, *y2;



int Ggetarray[objectname](obj, array)
GT_OBJECT obj;
struct garray[objectname] *array;

GT_OBJECT obj    INPUT: handle of a the graphics object.

struct garray[objectname] *array

                       INPUT: pointer to a garray[objectname]
                        structure.
                       RETURN: object values are assigned to
                        the corresponding structure entries.

      Returns size of the array in bytes.
```


DEFINITION

An arc is a section of an ellipse. It is defined by the semi-major and semi-minor axis of the ellipse (commonly denoted by 'a' and 'b' respectively), the center of the ellipse (xc, yc), and the start and end angle of the section of ellipse constituting the arc. Angles, measured in degrees, extend counterclockwise from the center of the ellipse where 0 degrees coincides with the right (EAST) direction.

TYPE

GO_ARC
GO_CARC

ATTRIBUTES

GARG_LWIDTH

Specifies the arc's width in world coordinates. The 'fat arc' actually becomes the original arc with linewidth/2 lines on each side of it.

GARG_COLOR

Specifies the arc's color (the arc's edge color when there is a linewidth).

GARG_FILL

Specifies how arcs and fat arcs are filled (if at all). Filled arcs which are not fat (i.e. linewidth is zero) are triangular 'pie slices' and if the fill color (GARG_FILLCOLOR) is G_TRANSPARENT the arcs are 'pie slices' with only the edges drawn.

GARG_WRITEMODE

Specifies how the arcs pixels are written to the video buffer.

GARG_HIDDEN

Specifies whether the arc is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Garc(xc, yc, a, b, start, end)
GT_COORD xc, yc, a, b;
GT_ANGLE start, end;

GT_OBJECT Gcarc(xc, yc, a, b, start, end)
GT_COORD xc, yc, a, b;
GT_ANGLE start, end;
```

Creates a arc primitive graphics object in the currently open segment. The primitive inherits the attributes found in the currently open context. A handle of the created object is returned to the caller.
Gcarc() creates a compiled arc primitive graphics object.

```
GT_OBJECT Gexecarc(values)
struct garrayarc *values;

GT_OBJECT Gexeccarc(values)
struct garrayarc *values;
```

Takes as input an address of a garrayarc structure and the creates the corresponding arc primitive graphics object.

*Destroy

```
void Gdelete(arc);
GT_OBJECT arc;
```

*Inquire

```
      void Ginqarc(arcobj, xc, yc, a, b, start, end)
      GT_OBJECT arcobj;
      GT_COORD *xc, *yc, *a, *b;/* RETURNED */
      GT_ANGLE *start, *end;/* RETURNED */

      int Ggetarrayarc(arcobj, array)
      GT_OBJECT arcobj;
      struct garrayarc *array;/* RETURNED */

            The size of the array buffer (i.e. struct garrayarc)
            is returned.

      Takes the input arc primitive graphics object and fills out the
      array buffer with values which are equivalent to the parameters
      which could be passed to Garc() to create a copy
      of the object. Also, the array returned from Ggetarrayarc can be
      passed to Gexecarc or Gexeccarc to create the a copy of the
      object. There is no attempt to verify that the input object is
      really a arc object.
```

*Modify

```
      void Gmodarc(obj, xc, yc, a, b, start, end)
      GT_OBJECT obj;
      GT_COORD xc, yc, a, b;
      GT_ANGLEstart, end;

      Modifies the given primitive graphics object with the given
      parameters to be equivalent to an object created with those
      same parameters (i.e. changes the arcs's center, size, and
      angles to the new values).

      All 'Standard Object Operations' are valid. Refer to the
      particular operation for descriptions in the 'Standard Object
      Operations' section.
```

*Draw

```
      void Gdraw(arc)
      GT_OBJECT arc;

      Renders the primitive graphics object using the current
      open picture to determine the world to device transform,
      the clipping bounds, and the window and device to draw to.

      void Gdrawarc(ctx, xc, yc, a, b, start, end)
      GT_CTX ctx;
      GT_COORD xc, yc, a, b;
      GT_ANGLE start, end;

      Similar to the create function, Garc, but immediately renders
      the arc, does not create a graphics object, and uses the
```

supplied context 'ctx' rather than the currently open context. The primitive is drawn using the open picture's world/viewport transformations and clipping boundaries.

CHILD OBJECTS

None.

PARENT OBJECTS

Segments
Calls

EXAMPLES

BUGS

SEE ALSO

Primitives, Context, Segments, Calls,

DEFINITION

The call primitive is a combination translation, rotation and scale operation applied to a segment or primitive. Drawing the call primitive causes the segment or primitive to be drawn (after the translation and/or rotation and/or scale is applied to it). It is usually used when there is an object that is to be displayed at multiple locations and orientations. In this case the top level segment contains multiple call primitives pointing to this popular segment.

TYPE

GO_CALL
GO_CCALL

ATTRIBUTES

GARG_HIDDEN

Specifies whether the segment is visible or not.

OPERATIONS

*Create

```
      GT_OBJECT Gcall(seg, tx, ty, angle, scale)
      GT_OBJECT seg;
      GT_COORD tx, ty;
      GT_ANGLE angle;
      GT_SCALE scale;

      GT_OBJECT Gccall(seg, tx, ty, angle, scale)
      GT_OBJECT seg;
      GT_COORD tx, ty;
      GT_ANGLE angle;
      GT_SCALE scale;

      Creates a call primitive graphics object in the currently
      open segment. The primitive inherits the attributes found in the
      currently open context.  A handle of the created object is
      returned to the caller.
      Gccall() creates a compiled call primitive graphics object.

      GT_OBJECT Gexeccall(values)
      struct garraycall *values;

      GT_OBJECT Gexecccall(values)
      struct garraycall *values;

      Takes as input an address of a garraycall structure and the
      creates the corresponding call primitive graphics object.

*Destroy

      void Gdelete(call);
      GT_OBJECT call;

*Inquire

      void Ginqcall(obj, seg, tx, ty, angle, scale)
      GT_OBJECT obj;
      GT_OBJECT *seg;         /* RETURNED */
      GT_COORD *tx, *ty;/* RETURNED */
      GT_ANGLE *angle;  /* RETURNED */
      GT_SCALE *scale;  /* RETURNED */

      int Ggetarraycall(obj, array)
      GT_OBJECT obj;
      struct garraycall *array;/* RETURNED */

            The size of the array buffer (i.e. struct garraycall)
            is returned.

      Takes the input call primitive graphics object and returns
      the corresponding values which are equivalent to the parameters
      which could be passed to Gcall() to create a copy
      of the object. Also, the array returned from Ggetarraycall can be
      passed to Gexeccall or Gexecccall to create the a copy of
      the object. There is no attempt to verify that the input object
```

is really a call object.

*Modify

```
void Gmodcall(icall, ...)
GT_OBJECT icall;
```

Modifies the given primitive graphics object with the given parameters to be equivalent to an object created with those same parameters.

Valid keywords:

GARG_SEGMENT

All 'Standard Object Operations' are valid. Refer to the particular operation for descriptions in the 'Standard Object Operations' section.

*Draw

```
void Gdraw(call)
GT_OBJECT call;
```

Renders the primitive graphics object using the current open picture to determine the world to device transform, the clipping bounds, and the window and device to draw to.

CHILD OBJECTS

Segments and Primitives.

PARENT OBJECTS

Segments
Calls

EXAMPLES

BUGS

Scale is not implemented at this time.
Gmodcall takes modifys only the segment, Gtranslate and Grotate must be used to modify its other values.

SEE ALSO

Primitives, Segments,

DEFINITION

An ellipse is a standard conic section. It is defined by the semi-major and semi-minor axis of the ellipse (commonly denoted by 'a' and 'b' respectively), the center of the ellipse (xc, yc). Only ellipses with orthogonal axis are supported at this time.

TYPE

GO_ELLIPSE
GO_CELLIPSE

ATTRIBUTES

GARG_LWIDTH

Specifies the ellipse's width in world coordinates. The 'fat ellipse' actually becomes the original ellipse with linewidth/2 lines on each side of it.

GARG_COLOR

Specifies the ellipse's color (the ellipse's edge color when there is a linewidth).

GARG_FILL

Specifies how ellipses and fat ellipses are filled (if at all).

GARG_WRITEMODE

Specifies how the ellipses pixels are written to the video buffer.

GARG_HIDDEN

Specifies whether the ellipse is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Gellipse(xc, yc, a, b)
GT_COORD xc, yc, a, b;

GT_OBJECT Gcellipse(xc, yc, a, b)
GT_COORD xc, yc, a, b;
```

Creates a ellipse primitive graphics object in the currently open segment. The primitive inherits the attributes found in the currently open context. A handle of the created object is

```
            returned to the caller.
            Gcellipse() creates a compiled ellipse primitive graphics object.

            GT_OBJECT Gexecellipse(values)
            struct garrayellipse *values;

            GT_OBJECT Gexeccellipse(values)
            struct garrayellipse *values;

            Takes as input an address of a garrayellipse structure and the
            creates the corresponding ellipse primitive graphics object.

      *Destroy

            void Gdelete(ellipse);
            GT_OBJECT ellipse;

      *Inquire

            void Ginqellipse(ellipseobj, xc, yc, a, b)
            GT_OBJECT ellipseobj;
            GT_COORD *xc, *yc, *a, *b;/* RETURNED */

            int Ggetarrayellipse(ellipseobj, array)
            GT_OBJECT ellipseobj;
            struct garrayellipse *array;/* RETURNED */

                  The size of the array buffer (i.e. struct garrayellipse)
                  is returned.

            Takes the input ellipse primitive graphics object and returns the
            corresponding values which are equivalent to the parameters
            which could be passed to Gellipse() to create a copy
            of the object. Also, the array returned from Ggetarrayellipse can
be
            passed to Gexecellipse or Gexeccellipse to create the a copy of
the
            object. There is no attempt to verify that the input object is
            really a ellipse object.

      *Modify

            void Gmodellipse(obj, xc, yc, a, b)
            GT_OBJECT obj;
            GT_COORD xc, yc, a, b;

            Modifies the given primitive graphics object with the given
            parameters to be equivalent to an object created with those
            same parameters (i.e. changes the ellipse's center, and size
            to the new values).

            All 'Standard Object Operations' are valid. Refer to the
            particular operation for descriptions in the 'Standard Object
            Operations' section.
```

*Draw

```
void Gdraw(ellipse)
GT_OBJECT ellipse;

Renders the primitive graphics object using the current
open picture to determine the world to device transform,
the clipping bounds, and the window and device to draw to.

void Gdrawellipse(ctx, xc, yc, a, b)
GT_CTX ctx;
GT_COORD xc, yc, a, b;

Similar to the create function, Gellipse, but immediately renders
the ellipse, does not create a graphics object, and uses the
supplied context 'ctx' rather than the currently open context.
The primitive is drawn using the open picture's world/viewport
transformations and clipping boundaries.

void (*Gr_ellipse)(mask, xc, yc, a, b)
short mask;
GT_DCOORD xc, yc, a, b;
```

CHILD OBJECTS

None.

PARENT OBJECTS

Segments
Calls

EXAMPLES

BUGS

SEE ALSO

Primitives, Context, Segments, Calls,

DEFINITION

A line is a visual connection between two points. It is defined by a start and end coordinate point. Point one (x1, y1) and point two (x2, y2) are in world coordinates.

TYPE

GO_LINE
GO_CLINE

ATTRIBUTES

GARG_LWIDTH

Specifies the line's width in world coordinates. The 'wide line' actually becomes the original line with linewidth/2 lines on each side of it. This is one way to generate a orthogonal rectangle rotated at an angle.

GARG_COLOR

Specifies the line's color (the line's edge color when there is a linewidth).

GARG_FILL

Specifies how wide lines are filled (if at all).

GARG_WRITEMODE

Specifies how the lines pixels are written to the video buffer.

GARG_HIDDEN

Specifies whether the line is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Gline(x1, y1, x2, y2)
GT_COORD x1, y1, x2, y2;

GT_OBJECT Gcline(x1, y1, x2, y2)
GT_COORD x1, y1, x2, y2;
```

Creates a line primitive graphics object in the currently open segment. The primitive inherits the attributes found in the currently open context. A handle of the created object is returned to the caller.
Gcline() creates a compiled line primitive graphics object.

```
GT_OBJECT Gexecline(values)
struct garrayline *values;

GT_OBJECT Gexeccline(values)
struct garrayline *values;
```

Takes as input an address of a garrayline structure and the creates the corresponding line primitive graphics object.

```
*Destroy

      void Gdelete(line);
      GT_OBJECT line;
*Inquire

      void Ginqline(lineobj, x1, y1, x2, y2)
      GT_OBJECT lineobj;
      GT_COORD *x1, *y1, *x2, *y2;/* RETURNED */

      int Ggetarrayline(lineobj, array)
      GT_OBJECT lineobj;
      struct garrayline *array;/* RETURNED */

            The size of the array buffer (i.e. struct garrayline)
            is returned.

      Takes the input line primitive graphics object and returns the
      values of the world coordinate start and end points. These
      values are equivalent to the parameters passed to Gline()
      which will then create a copy of the object. Also, the array
      can be passed to Gexecline or Gexeccline to create the a copy
      of the object. There is no attempt to verify that the input
      object is really a line object.


*Modify

      void Gmodline(obj, x1, y1, x2, y2)
      GT_OBJECT obj;
      GT_COORD x1, y1, x2, y2;

      Modifies the given primitive graphics object with the given
      parameters to be equivalent to an object created with those
      same parameters (i.e. changes the line's endpoints to the
      new values).

      All 'Standard Object Operations' are valid. Refer to the
      particular operation for descriptions in the 'Standard Object
      Operations' section.

*Draw

      void Gdraw(line)
      GT_OBJECT line;

      Renders the primitive graphics object using the current
      open picture to determine the world to device transform,
      the clipping bounds, and the window and device to draw to.

      void Gdrawline(ctx, x1, y1, x2, y2)
      GT_CTX ctx;
      GT_COORD x1, y1, x2, y2;
```

Takes the input coordinates as representing the endpoints of a line and renders the line using the supplied context ctx. The line is drawn using the open picture's world/viewport transformations and clipping boundaries.

```
void (*Gr_aline)(x1, y1, x2, y2)
GT_DCOORD x1, y1, x2, y2;
```

CHILD OBJECTS

None.

PARENT OBJECTS

Segments
Calls

EXAMPLES

BUGS

SEE ALSO

Primitives, Context, Segments, Calls,

DEFINITION

The polygon primitive is used to display a group of polylines and arcs. It is defined by a pointer to a list of polylines and a pointer to a list of arcs, either of which may be NULL.
The 'linelist' parameter points to a list of polylines of form:

```
GT_COORD number_of_lines;
GT_COORD pt1_x, pt1_y;
GT_COORD pt2_x, pt2_y;
GT_COORD pt3_x, pt3_y;
...
GT_COORD number_of_lines_polyline_#2;
GT_COORD pt1_x, pt1_y;
GT_COORD pt2_x, pt2_y;
GT_COORD pt3_x, pt3_y;
...
...
GT_COORD 0;/* end of polylines */
```

Similarly, the 'arclist' parameter points to a list of arc data of form:

```
GT_COORD number_of_arcs;
GT_COORD xc, yc, a, b, start, end;/* arc #1 */
GT_COORD xc, yc, a, b, start, end;/* arc #2 */
...
```

TYPE

GO_POLYGON
GO_CPOLYGON

ATTRIBUTES

GARG_LWIDTH

Specifies the lines' and arcs' width in world coordinates and behaves exactly like individual lines and arcs.

GARG_COLOR

Specifies the lines' and arcs' color (the lines' and arcs' edge color when there is a linewidth).

GARG_FILL

Specifies how fat lines and arcs are filled (if at all). Also used to fill horizontal slices of the polygon formed by the polylines and arcs (whether it is closed or not) when linewidth is zero. This can be used to create areas with unfilled 'holes' in them.

GARG_WRITEMODE

Specifies how the lines' and arcs' pixels are written to the video buffer.

GARG_HIDDEN

Specifies whether the polygon is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Gpolygon(linelist, arclist)
GT_COORD *linelist;
GT_COORD *arclist;

GT_OBJECT Gcpolygon(linelist, arclist)
GT_COORD *linelist;
```

```
            GT_COORD *arclist;

            Creates a polygon primitive graphics object in the currently
            open segment. The primitive inherits the attributes found in the
            currently open context.  A handle of the created object is
            returned to the caller.
            Gcpolygon() creates a compiled polygon primitive graphics object.

            GT_OBJECT Gexecpolygon(values)
            struct garraypolygon *values;

            GT_OBJECT Gexeccpolygon(values)
            struct garraypolygon *values;

            Takes as input an address of a garraypolygon structure and the
            creates the corresponding polygon primitive graphics object.

      *Destroy

            void Gdelete(polygon);
            GT_OBJECT polygon;

      *Inquire

            void Ginqpolygon(polygonobj, linelist, arclist)
            GT_OBJECT polygonobj;
            GT_COORD **linelist;/* RETURNED */
            GT_COORD **arclist;/* RETURNED */

            int Ggetarraypolygon(polygonobj, array)
            GT_OBJECT polygonobj;
            struct garraypolygon *array;/* RETURNED */

                  The size of the array buffer (i.e. struct garraypolygon)
                  is returned.

            Takes the input polygon primitive graphics object and returns
            the corresponding values which are equivalent to the parameters
            which could be passed to Gpolygon() to create a copy
            of the object. Also, the array returned from Ggetarraypolygon can
be
            passed to Gexecpolygon or Gexeccpolygon to create the a copy of
            the object. There is no attempt to verify that the input object
            is really a polygon object.

      *Modify

            void Gmodpolygon(obj, linelist, arclist)
            GT_OBJECT obj;
            GT_COORD *linelist;
            GT_COORD *arclist;

            Modifies the given primitive graphics object with the given
            parameters to be equivalent to an object created with those
```

same parameters (i.e. changes the polygons's list of polylines and arcs to the new lists).

All 'Standard Object Operations' are valid. Refer to the particular operation for descriptions in the 'Standard Object Operations' section.

*Draw

```
void Gdraw(polygon)
GT_OBJECT polygon;
```

Renders the primitive graphics object using the current open picture to determine the world to device transform, the clipping bounds, and the window and device to draw to.

```
void Gdrawpolygon(ctx, linelist, arclist)
GT_CTX ctx;
GT_COORD *linelist;
GT_COORD *arclist;
```

Similar to the create function, Gpolygon, but immediately renders the polygon, does not create a graphics object, and uses the supplied context 'ctx' rather than the currently open context. The primitive is drawn using the open picture's world/viewport transformations and clipping boundaries.

CHILD OBJECTS

None.

PARENT OBJECTS

Segments
Calls

EXAMPLES

BUGS

Gmodpolygon is NOT IMPLEMENTED.

SEE ALSO

Primitives, Context, Segments, Calls,

DEFINITION

The polyline primitive is used to display lines which are connected end-to-end. It is defined by a pointer to a list of end points and by the number of lines defined by the list.

TYPE

GO_PLINE
GO_CPLINE

ATTRIBUTES

GARG_LWIDTH

Specifies the lines' width in world coordinates. All the 'fat lines' actually become the original lines with linewidth/2 lines on each side of it.

GARG_COLOR

Specifies the lines' color (the lines' edge color when there is a linewidth).

GARG_FILL

Specifies how fat lines are filled (if at all). Also used to fill horizontal slices of the polygon formed by the polylines (whether it is closed or not) when linewidth is zero.

GARG_WRITEMODE

Specifies how the lines pixels are written to the video buffer.

GARG_HIDDEN

Specifies whether the polyline is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Gpline(ptlist, number_of_line_segments)
GT_COORD *ptlist;
int number_of_line_segments;

GT_OBJECT Gcpline(ptlist, number_of_line_segments)
GT_COORD *ptlist;
int number_of_line_segments;
```

Creates a pline primitive graphics object in the currently open segment. The primitive inherits the attributes found in the currently open context. A handle of the created object is

```
      returned to the caller.
      Gcpline() creates a compiled pline primitive graphics object.

      GT_OBJECT Gexecpline(values)
      struct garraypline *values;

      GT_OBJECT Gexeccpline(values)
      struct garraypline *values;

      Takes as input an address of a garraypline structure and the
      creates the corresponding pline primitive graphics object.

*Destroy

      void Gdelete(pline);
      GT_OBJECT pline;

*Inquire

      void Ginqpline(plineobj, ptlist, number_of_line_segments)
      GT_OBJECT plineobj;
      GT_COORD **ptlist;/* RETURNED */
      int *number_of_line_segments;/* RETURNED */

      int Ggetarraypline(plineobj, array)
      GT_OBJECT plineobj;
      struct garraypline *array;/* RETURNED */

            The size of the array buffer (i.e. struct garraypline)
            is returned.

      Takes the input pline primitive graphics object and returns the
      corresponding values which are equivalent to the parameters
      which could be passed to Gpline() to create a copy
      of the object. Also, the array returned from Ggetarraypline can be
      passed to Gexecpline or Gexeccpline to create the a copy of the
      object. There is no attempt to verify that the input object is
      really a pline object.

*Modify

      void Gmodpline(obj, coords, number)
      GT_OBJECT obj;
      GT_COORD *coords;
      int         number;/* number of line segments */

      Modifies the given primitive graphics object with the given
      parameters to be equivalent to an object created with those
      same parameters (i.e. changes the polylines's list of points
      to the new values).

      All 'Standard Object Operations' are valid. Refer to the
      particular operation for descriptions in the 'Standard Object
      Operations' section.
```

*Draw

```
void Gdraw(pline)
GT_OBJECT pline;
```

Renders the primitive graphics object using the current open picture to determine the world to device transform, the clipping bounds, and the window and device to draw to.

```
void Gdrawpline(ctx, ptlist, number_of_line_segments)
GT_CTX ctx;
GT_COORD *ptlist;
int number_of_line_segments;
```

Similar to the create function, Gpline, but immediately renders the pline, does not create a graphics object, and uses the supplied context 'ctx' rather than the currently open context. The primitive is drawn using the open picture's world/viewport transformations and clipping boundaries.

CHILD OBJECTS

None.

PARENT OBJECTS

Segments
Calls

EXAMPLES

BUGS

SEE ALSO

For groups of disjoint lines see polygons.
Primitives, Context, Segments, Calls,

DEFINITION

A rectangle is a group of four orthogonal lines. Rectangles are defined by the coordinate of the lower left hand corner and the coordinate of the upper right hand corner.

TYPE

GO_RECT
GO_CRECT

ATTRIBUTES

GARG_COLOR

Specifies the rectangle's edge color.

GARG_FILL

Specifies how the rectangle is filled (if at all).

GARG_WRITEMODE

Specifies how the rectangles pixels are written to the video buffer.

GARG_HIDDEN

Specifies whether the rectangle is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Grect(x1, y1, x2, y2)
GT_COORD x1, y1, x2, y2;

GT_OBJECT Gcrect(x1, y1, x2, y2)
GT_COORD x1, y1, x2, y2;
```

Creates a rectangle primitive graphics object in the currently open segment. The primitive inherits the attributes found in the currently open context. Point one specifies the lower left hand corner and point two specifies the upper right hand corner. The points are in world coordinates. A handle of the created object is returned to the caller.
Gcrectangle() creates a compiled rectangle primitive graphics object.

```
GT_OBJECT Gexecrect(values)
struct garrayrect *values;

GT_OBJECT Gexeccrect(values)
struct garrayrect *values;
```

Takes as input an address of a garrayrect structure and the creates the corresponding rectangle primitive graphics object.

*Destroy

```
      void Gdelete(rectangle);
      GT_OBJECT rectangle;

*Inquire

      void Ginqrect(rectangleobj, x1, y1, x2, y2)
      GT_OBJECT rectangle;
      GT_COORD *x1, *y1, *x2, *y2;/* RETURNED */

      int Ggetarrayrect(rectangleobj, array)
      GT_OBJECT rectangle;
      struct garrayrect *array;/* RETURNED */

            The size of the array buffer (i.e. struct garrayrect)
            is returned.

      Takes the input rectangle primitive graphics object and returns
      the values of the world coordinate start and end points. These
      values are equivalent to the parameters passed to Grect()
      which will then create a copy of the object. Also, the array
      can be passed to Gexecrect or Gexeccrect to create the a copy
      of the object. There is no attempt to verify that the input
      object is really a rect object.


*Modify

      void Gmodrect(obj, x1, y1, x2, y2)
      GT_OBJECT obj;
      GT_COORD x1, y1, x2, y2;

      Modifies the given primitive graphics object with the given
      parameters to be equivalent to an object created with those
      same parameters (i.e. changes the rectangles's lower-left-hand
      and upper-right-hand corners to the new values).

      All 'Standard Object Operations' are valid. Refer to the
      particular operation for descriptions in the 'Standard Object
      Operations' section.

*Draw

      void Gdraw(rectangle)
      GT_OBJECT rectangle;

      Renders the primitive graphics object using the current
      open picture to determine the world to device transform,
      the clipping bounds, and the window and device to draw to.

      void Gdrawrect(ctx, x1, y1, x2, y2)
      GT_CTX ctx;
      GT_COORD x1, y1, x2, y2;

      Takes the input coordinates as representing the lower left and
```

upper right corners of a rectangle and renders it using the supplied context ctx. The primitive is drawn using the open picture's world/viewport transformations and clipping boundaries.

CHILD OBJECTS

None.

PARENT OBJECTS

Segments
Calls

EXAMPLES

BUGS

SEE ALSO

Primitives, Context, Segments, Calls,

DEFINITION

The text primitive is used to display alphanumeric characters. It is defined by an ASCII text string and the coordinate of the lower left hand corner of the first character. Only stroke font characters may be rotated at this time.

TYPE

GO_TEXT
GO_CTEXT

ATTRIBUTES

GARG_THEIGHT

Specifies the height of the characters in world coordinates.

GARG_TWIDTH

Specifies the width of each character in world coordinates.

GARG_TSPACING

Specifies the spacing between the characters in world coordinates.

GARG_TFONT

Specifies which font is used when the text string is rendered. The default is font 0 which is a resident fixed size (8~14 pixels high by 8 pixels wide) bitmapped font.

GARG_COLOR

Specifies the text color.

GARG_WRITEMODE

Specifies how the text string pixels are written to the video buffer.

GARG_HIDDEN

Specifies whether the text is visible or not.

OPERATIONS

*Create

```
GT_OBJECT Gtext(x, y, string)
GT_COORD x, y;
char *string;

GT_OBJECT Gctext(x, y, string)
GT_COORD x, y;
char *string;
```

Creates a text primitive graphics object in the currently open segment. The primitive inherits the attributes found in the currently open context. A handle of the created object is returned to the caller.
Gctext() creates a compiled text primitive graphics object.

```
GT_OBJECT Gexectext(values)
struct garraytext *values;

GT_OBJECT Gexecctext(values)
struct garraytext *values;
```

Takes as input an address of a garraytext structure and the creates the corresponding text primitive graphics object.

*Destroy

```
void Gdelete(text);
GT_OBJECT text;
```

*Inquire

```
      void Ginqtext(textobj, x, y, str)
      GT_OBJECT textobj;
      GT_COORD *x, *y;  /* RETURNED */
      char **str;             /* RETURNED */

      int Ggetarraytext(textobj, array)
      GT_OBJECT textobj;
      struct garraytext *array;/* RETURNED */

            The size of the array buffer (i.e. struct garraytext)
            is returned.

      Takes the input text primitive graphics object and returns the
      corresponding values which are equivalent to the parameters
      which could be passed to Gtext() to create a copy
      of the object. Also, the array returned from Ggetarraytext can
      be passed to Gexectext or Gexecctext to create the a copy of the
      object. There is no attempt to verify that the input object is
      really a text object.

*Modify

      void Gmodtext(obj, x, y, str)
      GT_OBJECT obj;
      GT_COORD x, y;
      char  *str;

      Modifies the given primitive graphics object with the given
      parameters to be equivalent to an object created with those
      same parameters (i.e. changes the text's position and string
      to the new values).

      All 'Standard Object Operations' are valid. Refer to the
      particular operation for descriptions in the 'Standard Object
      Operations' section.

*Draw

      void Gdraw(text)
      GT_OBJECT text;

      Renders the primitive graphics object using the current
      open picture to determine the world to device transform,
      the clipping bounds, and the window and device to draw to.

      void Gdrawtext(ctx, x, y, string)
      GT_CTX ctx;
      GT_COORD x, y;
      char *string;

      Similar to the create function, Gtext, but immediately renders
      the text, does not create a graphics object, and uses the
      supplied context 'ctx' rather than the currently open context.
      The primitive is drawn using the open picture's world/viewport
```

transformations and clipping boundaries.

```
void (*Gr_putchr)(character, x, y)
char character;
GT_DCOORD x, y;
```

CHILD OBJECTS

None.

PARENT OBJECTS

Segments
Calls

EXAMPLES

BUGS

SEE ALSO

Primitives, Context, Segments, Calls,

DEFINITION

The userobj primitive is used by applications who want to add their own graphical objects. It is defined by an address of the objects private data and by the address of an array of pointers to functions which are standard for all PGL graphic objects. These functions are call on to provide the functionality common to all graphic objects.

TYPE

GO_USEROBJ

ATTRIBUTES

GARG_HIDDEN

Specifies whether the userobj is visible or not.

All other attributes are available to the userobj's functions and it is up to these functions as to how they interpret each function.

OPERATIONS

All 'Standard Object Operations' are valid. Refer to the particular operation for descriptions in the 'Standard Object Operations' section.

*Create

```
GT_OBJECT Guserobj(data, funcs)
char *data;
struct gmbrfuncs *funcs;
```

Creates a userobj primitive graphics object in the currently open segment. The primitive inherits the attributes found in the currently open context. A handle of the created object is returned to the caller.

The struct 'gmbrfuncs' is a structure of pointers to routines which handle the various duties an object in PGL is requested to perform (i.e. draw itself, delete itself, etc...). These pointers are set to NULL when a gmbrfuncs structure is created and it is up to the application replace any of these pointers with the addresses of functions which will perform the jobs the application wants the userobj to do. In C++ parlance, this is an abstract class and the application overrides any member functions it needs in order to define the userobj functionality.

```
struct gmbrfuncs *Gmbrfuncs(subtype)
GT_TYPE subtype;
```

Creates a struct gmbrfuncs array.

```
void Gsetmemfuncsforsubtype(mf, subtype)
struct gmbrfuncs *mf;
GT_TYPE subtype;
```

Registers a struct gmbrfuncs array as the methods for objects of type GO_USEROBJ and given subtype.

```
struct gmbrfuncs * Ggetmemfuncsforsubtype(subtype)
GT_TYPE subtype;
```

Inquires what the methods are for objects of type GO_USEROBJ and the given subtype.

```
void Gdeletememfuncs(mf)
struct gmbrfuncs *mf;
```

Delete the given struct gmbrfuncs.

```
GT_TYPE Ggenuniquesubtype()
```

Return a subtype that has not yet been used.

```
GT_TYPE Ggetsubtypeuserobj(obj)
```

```
            struct guserobj *obj;

            Return the subtype of a user object (invokes the
            getsubtype function assigned to the userobj, if
            any).

      GT_OBJECT Gexecuserobj(values)
      struct guserobjarray *values;

      Takes as input an address of a guserobjarray structure and if
      the application has overridden this funcion, calls the 'Gexec'
      funtion of the userobj with 'values' as a parameter.

*Destroy

      void Gdelete(userobj);
      GT_OBJECT userobj;

      Invokes the delete method assigned to the userobj, if any, then
      deletes the userobj.

*Inquire

      void Ginquserobj(obj, data, funcs)
      GT_OBJECT obj;
      char **data;            /* RETURNED */
      struct gmbrfuncs **funcs;/* RETURNED */

      Takes the input userobj primitive graphics object and returns
      the corresponding values which are equivalent to the parameters
      which could be passed to Guserobj() to create a copy
      of the object. Also, the array returned from Ggetarraytext can
      be passed to Gexectext or Gexecctext to create the a copy of the
      object. There is no attempt to verify that the input object is
      really a userobj object.

*Modify

      void Gmoduserobj(obj, data, funcs)
      GT_OBJECT obj;
      char *data;
      struct gmbrfuncs *funcs;

      Modifies the given primitive graphics object with the given
      parameters to be equivalent to an object created with those
      same parameters (i.e. changes the userobjs's data and methods
      to the new values).

      void Goverridememberfunction(objtype, functionid, newfnaddr)
      GT_TYPE objtype;
      GT_TYPE functionid;
      GT_FNPTR newfnaddr;

      For userobjects of the given subtype, replace the standard
```

member function with the given one. Function ids defined at this time are:

GF_WRITEC
GF_GETOBJECTDESCRIPTION

*Draw

```
void Gdraw(userobj)
GT_OBJECT userobj;
```

Invokes the draw function assigned to the userobj, if any, to render the primitive graphics object using the current open picture to determine the world to device transform, the clipping bounds, and the window and device to draw to.

```
void Gdrawuserobj(ctx, data, funcs)
GT_CTX ctx;
char *data;
struct gmbrfuncs *funcs;
```

Similar to the create function, Guserobj, but immediately renders the userobj, does not create a graphics object, and uses the supplied context 'ctx' rather than the currently open context. The primitive is drawn using the open picture's world/viewport transformations and clipping boundaries. Again, this function does nothing unless a draw function has been assigned the object.

```
void Gundrawuserobj(obj)
struct guserobj *obj;
```

Invokes the undraw function assigned to the userobj, if any.

```
void Grecalcextremauserobj(obj)
struct guserobj *obj;
```

Called by the application's userobj functions to inform PGL that the userobj extrema has changed and that any extrema kept internally by PGL must now be updated (by a call to the function assigned to be the userobject's getextrema function.

CHILD OBJECTS

None.

PARENT OBJECTS

Segments

EXAMPLES

BUGS

SEE ALSO

Primitives, Context, Segments, Ginqdraw, Gmoddraw, Gmodtransform, Ginqtransform.

PGL OPERATIONS

Gtranslate(obj, dx, dy) position the object at a distance from where it was.

Gposition(obj, x, y) position an objects's reference point.

Grotate(obj, angle) rotates an object (currently only segments and instances).

Gscale(obj, newxdim, oldxdim, newydim, oldydim) scale the size of the object.

Gresize(obj, xext, yext) set the size of the object.

Gdraw(obj) draw object (segments and primitives are transformed according

Ghide(obj, flag) hide/unhide object.

Gsetclip(win, wxmin, wymin, wxmax, wymax) set clip bounds for subsequent draws.

Gunsetclip() restore clip bounds to open windows viewport.

Gsetdeviceclip(dxmin, dymin, dxmax, dymax) set clip bounds for subsequent draws.

Gunsetdeviceclip() restore clip bounds to open windows viewport.

Ggetextrema(obj, bounds) returns dimensional extrema of an object.

G_TYPE Ggettype(obj) inquire type of an object.

Gopen(obj) makes obj the currently open object (screen, window, segment).

Gclose(obj) does nothing.

GT_OBJECT Ggetopen(type) returns current open object of given type.

Gpushopen(obj) saves current open object(of same type) and opens obj.

Gpopopen(obj) makes the previously saved object of same type the open object.

Ggetarray(obj, array) fills out array with data representing object.

GT_OBJECT Gexecarray(array) create object from data in array.

GT_OBJECT Gapplyarray(object, array) modify object using data in array.

G_OBJECT Gnext(obj) return next object in structure in which obj resides.

G_OBJECT Gprev(obj) return previous object in structure in which obj resides.

G_OBJECT Gfirst(obj) return first object of structure in which obj resides.

G_OBJECT Glast(obj) return last object of structure in which obj resides.

G_OBJECT Gfirstelement(obj) return first object in substructure of obj.

G_OBJECT Glastelement(obj) return last object in substructure of obj.

G_OBJECT Gparent(obj) return handle of parent of obj (NULL if none).

Gexcise(obj) detach object from it's parent.

Ginsert(obj, obefore) insert obj into structure right before obefore

Gappend(obj, destobj) append obj to end of destobj, making destobj the parent.

Gpreappend(obj, destobj) append obj to beginning of destobj, the parent.

Gdelete(obj) delete obj from structure.

G_OBJECT Gcopy(obj) make copy of object and contents and return copies' handle.

Gpurge(obj) delete all elements of substructures of obj but not obj itself.

Rendering Management

to currently open window).

Gtranslate PGL OPERATIONS Gtranslate

DEFINITION

Translates a user specified graphic object in the (x, y) coordinate plane. This may change the inquireable values of the object. The resultant object is not checked whether it is still in the coordinate space (i.e. if it overflowed).

OPERATION

```
void Gtranslate(obj, dx, dy)
GT_OBJECT obj;
GT_COORD dx, dy;
```

PRIMITIVES

Adjusts the primitives internal coordinate description by the world coordinate translation (dx, dy).

SEGMENTS

Adjusts the segments position by adding the world coordinate translation (dx, dy) to the segment elements (transx, transy).

WINDOWS

Adjusts the windows screen position by the screen coordinate translation (dx, dy).

VIEWS

Adjusts the view by the world coordinate translation (dx, dy) by adding these values to the view's world coordinate parameters (this is the same as pan (i.e. Gpanview)).

EXAMPLES

BUGS

The graphic objects GO_DEVICE, GO_PICTURE and GO_SYMBOLICPICTURE are not supported and this function will return without doing anything when these are found.
The coordinates input must be of type GT_COORD but these are interpreted as type GT_SCOORD to translate a window object.

SEE ALSO

Gposition,

DEFINITION

Translates a user specified graphic object in the (x, y) coordinate plane so that it's reference point lies at the specified position. This may change the inquireable values of the object. The resultant object is not checked whether it is still in the coordinate space (i.e. if it overflowed).

OPERATION

```
void Gposition(obj, x, y)
GT_OBJECT obj;
GT_COORD x, y;
```

PRIMITIVES

Adjusts the primitives internal coordinate description such that the world coordinate reference point lies at (x, y). See the specific primitive documentation for information on each one's reference point.

SEGMENTS

Adjusts the segments position by adding a world coordinate translation to the segment elements (transx, transy) such that the segment's reference point lies at (x, y). The segment reference point is the world coordinate of the lower left hand corner of the segment's extrema.

WINDOWS

Adjusts the windows screen position such that the screen's screen coordinate reference point lies at (x, y). The screen reference point is the screen coordinate of it's lower left hand corner.

VIEWS

Adjusts the view by a world coordinate translation such that the view's reference point lies at (x, y). The view reference point is the world coordinate of it's lower left hand corner.

EXAMPLES

BUGS

The graphic objects GO_DEVICE, GO_PICTURE and GO_SYMBOLICPICTURE are not supported and this function will return without doing anything when these are found.
The coordinates input must be of type GT_COORD but these are interpreted as type GT_SCOORD to position a window object.

SEE ALSO

Gtranslate,

DEFINITION

Rotates a user specified graphic object in the (x, y) coordinate plane so that it's reference point lies at the specified position. This may change the inquireable values of the object. The resultant object is not checked whether it is still in the coordinate space (i.e. if it overflowed).

OPERATION

```
void Grotate(obj, angle)
GT_OBJECT obj;
GT_ANGLE angle;
```

PRIMITIVES

Adjusts the call primitives internal angle element such that the primitive or segment that the primitive calls are rotated to a new angle. The new angle is 'angle' degrees added to the old one. This does not change the actual graphic elements the call invokes in any way.

SEGMENTS

Adjusts the segments orientation by adding the specified angle to the internal segment 'angle' element. All objects in the segment will be displayed at an additional rotation of 'angle' degrees when the segment is drawn. This does not change the actual graphic elements of the segment in any way.

EXAMPLES

BUGS

The graphic objects GO_WINDOW, GO_VIEW, GO_DEVICE, GO_PICTURE and GO_SYMBOLICPICTURE and all primitives besides the call primitive are not supported and this function will return without doing anything when these are found.

SEE ALSO

Gcall, Gsegment

DEFINITION

Scales the given object by the given factors. This changes the actual internal data of the object.

PURPOSE

To provide a method of scaling objects hiding the fact that each object type must use different algorithms to accomplish this.

OPERATION

```
int Gscale(obj, newxdim, oldxdim, newydim, oldydim)
GT_OBJECT obj;
GT_COORD newxdim, oldxdim;
```

GT_COORD newydim, oldydim;

Non-zero is returned if this operation fails (i.e. because the operation is not defined for the given object).
The scale factor along the X-axis is newxdim/oldxdim. Similarly the Y-axis scale factor is newydim/oldydim. For each coordinate, the scale factors are multiplied by the distance of the coordinate from the scaling center.

PRIMITIVES

The scaling center is the center of the primitive extrema.

SEGMENTS

The scaling center is the center of the segment extrema. All primitives in the segment are scaled around this point.

WINDOWS

This operation is undefined for this object type.

VIEWS

This operation is undefined for this object type.

PICTURES

This operation is undefined for this object type.

SYMBOLIC PICTURES

This operation is undefined for this object type.

DEVICES

This operation is undefined for this object type.

EXAMPLES

```
static void play_with_object_size(object)
GT_OBJECT object;
      {
      GT_EXTREMA bounds;
      Ggetextrem(object, &bounds);


      /* make the object be 100 world coordinates tall
                  and 200 world coordinates wide */
      Gscale(object,
            /* newxdim, oldxdim */
            (GT_COORD )100, bounds.xmax - bounds.xmin,
            /* newydim, oldydim */
            (GT_COORD )200, bounds.ymax - bounds.ymin);
```

```
      /* double the size of the object */
      Gscale(object, (GT_COORD )2, (GT_COORD )1, (GT_COORD )2, (GT_COORD )1):
      }
```

BUGS

SEE ALSO

Gresize, Gtranslate, Grotate, Gposition.

DEFINITION

Resizes the given object to be the given size.

PURPOSE

To provide a method of resizing objects hiding the fact that each object type must use different algorithms to accomplish this.

OPERATION

```
int Gresize(obj, xext, yext)
GT_OBJECT obj;
GT_COORD xext, yext;
```

Non-zero is returned if this operation fails (i.e. because the operation is not defined for the given object).

PRIMITIVES

The primitive is resized to have the given extrema, keeping the reference point at the same location.

SEGMENTS

All primitives in the segment are resized to have the given extrema. As if the Gresize call had been made for each primitive separately.

WINDOWS

This operation is undefined for this object type.

VIEWS

This operation is undefined for this object type.

PICTURES

This operation is undefined for this object type.

SYMBOLIC PICTURES

This operation is undefined for this object type.

DEVICES

This operation is undefined for this object type.

EXAMPLES

BUGS

SEE ALSO

Gscale, Gtranslate, Grotate, Gposition.

PGL DRAW OPERATIONS

The following are rendering routines that accept a context and world coordinates and then draw to the currently open window.

```
void Gdrawrectangle(ctx, x1, y1, x2, y2)
void Gdrawtext(ctx, x, y, string, len)
void Gdrawline(ctx, x1, y1, x2, y2)
void Gdraw3line(ctx, x1, y1, z1, x2, y2, z2)
void Gdrawpline(ctx, array, number)
void Gdraw3pline(ctx, array, number)
void Gdrawelipse(ctx, xc, yc, a, b)
void Gdrawarc(ctx, xc, yc, a, b, start, end)
void Gdrawpolygon(ctx, linelist, arclist)
```

The following are rendering routines that accept device coordinates and will do little or no clipping and directly call the lowest level graphics (or resident rendering system).

```
void  (*Gr_aline)();
void  (*Gr_putchr)();
void  (*Gr_elipse)();
void  (*Gr_clear)();
void  (*Gr_pixel)();
void  (*Gr_fline)();
void    (*Gr_fclear)();
void    (*Gr_fpattern)();
void  (*Gr_rop)();
void  (*Gr_cursor)();
void  (*Gr_setwritemode)();
void  (*Gr_setcolor)();
void  (*Gr_setclipbounds)();
```

See the description of the particular primitive for more information.

DEFINITION

Renders a user specified graphic object.

OPERATION

```
void Gdraw(obj)
GT_OBJECT obj;
```

PRIMITIVES

Renders the primitive using the currently open picture's world/viewport transformations, clipping boundaries, window and device. If the primitive has the hidden attribute set to TRUE in it's graphic context then it is not drawn.

SEGMENTS

Renders the primitives in the segment using the currently open picture's world/viewport transformations, clipping boundaries, window and device. If there is a translation and or rotation associated with the segment, these are applied to each of it's primitives. If the primitive has the hidden attribute set to TRUE in it's graphic context then it is not drawn.

WINDOWS

Renders each picture that has been associated with the window. (or conversely, renders each picture that has as it's window element the input window to Gdraw). Note that this may cause graphics to be drawn on multiple devices.

VIEWS

Renders each picture that has been associated with the view. (or conversely, renders each picture that has as it's view element the input view to Gdraw). Note that this may cause graphics to be drawn in multiple windows on multiple devices.

PICTURES

Renders the segment data based on the world/viewport transformation derived from the picture's 'view' element to the picture's 'window' element on the picture's device element. The data is clipped to the 'view's world coordinate boundary. The picture's GARG_FILL, GARG_PATTERN attributes in it's graphic context determine the pictures background color and fill pattern (if any).

SYMBOLIC PICTURES

Renders each associated picture.

DEVICES

Not supported at this time.....

SUPPORT FUNCTIONS

```
void Gmoddraw(va_alist)
va_dcl
```

Modifes the current draw state.

Allowable keywords:

```
GARG_TRANSLATION(GT_D32COORD )x translation,
                 (GT_D32COORD )y translation.
                 Sets the device translation
                 in device coordinates.

GARG_DEVICECLIPRECT(GT_D32COORD )xmin,
                 (GT_D32COORD )ymin,
                 (GT_D32COORD )xmax,
                 (GT_D32COORD )ymax
                 Sets the clipping rectangle
                 in device (16:16) coordinates.

GARG_CLIPRECT(GT_COORD )xmin,
                 (GT_COORD )ymin,
                 (GT_COORD )xmax,
                 (GT_COORD )ymax
                 Sets the clipping rectangle
                 in world coordinates.
```

```
void Ginqdraw(va_alist)
va_dcl
```

Inquires the current draw state and returns the values requested. Valid 'GARG_' keywords are the same as those for Gdrawmodify.

```
void Gdrawpushstate(state)
char **state;
```

Saves the current draw state and returns its handle.

```
void Gdrawpopstate(state)
char *state;
```

Restores the given graphics draw state to the state specified by the given handle.

```
void Gmodtransform(va_alist)
va_dcl
```

Modifies the current drawing transformation function.

Allowable keywords:

```
GARG_DRAWTRANSFORMATION_PROCS(GT_D32FNPTR)transx,
                             (GT_D32FNPTR)transy
                             The routines which
                             take a world coord
                             (GT_COORD) and transform
                             it into a device coord
                             (GT_D32COORD).
                             If NULL, a 1-to-1 world
                             to device transformation
                             is installed.
```

```
void Ginqtransform(va_alist)
va_dcl
```

Inquires the current draw transform and returns the values requested. Valid 'GARG_' keywords are the same as those for Gmodtransform.

EXAMPLES

BUGS

SEE ALSO

Ghide, Gsetclip, Gunsetclip Gsetdeviceclip, Gunsetdeviceclip

Ghide PGL DRAW OPERATIONS
Ghide

DEFINITION

Hides/unhides a user specified graphic object. The displayed object will be undrawn/drawn and it's hidden attribute set/cleared. The method of undrawing is to set the objects hidden attribute in it's graphic context to TRUE and then draw the graphics objects that were behind and within it's extrema.

OPERATION

```
void Ghide(obj, flag)
GT_OBJECT obj;
int flag;
```

PRIMITIVES

Hides/unhides primitive if flag TRUE/FALSE. Assumes the primitive exists in the currently open picture, which is used to redraw background (to hide) and to draw the primitive (unhide).

SEGMENTS

Hides/unhides segment if flag TRUE/FALSE. Assumes the segment exists in the currently open picture, which is used to redraw background (to hide) and to draw the segment (unhide).

WINDOWS

Hides/unhides window if flag TRUE/FALSE.

VIEWS

Hides/unhides view if flag TRUE/FALSE. To hide view each picture associated with the view has it's window element redrawn to clip bounds set to the view's extrema.

PICTURES

Hides/unhides picture if flag TRUE/FALSE. To hide picture it's window element is redrawn to clip bounds set to the pictures extrema.

SYMBOLIC PICTURES

Hides/unhides each picture if flag TRUE/FALSE.

DEVICES

Not supported at this time.....

EXAMPLES

BUGS

The graphic objects GO_DEVICE is not supported and this function will return without doing anything when this is found.

SEE ALSO

Gdraw, Gsetclip, Gunsetclip Gsetdeviceclip, Gunsetdeviceclip

DEFINITION

Specifies a clipping rectangle to be used in addition to the world bounds of the view in the currently open picture. This additional clipping criteria is in effect until the associated function 'Gunsetclip' is called.

OPERATION

```
void Gsetclip(picture, wxmin, wymin, wxmax, wymax)
GT_OBJECT picture;
GT_COORD wxmin, wymin, wxmax, wymax;
```

The parameters (wxmin, wymin, wxmax, wymax) are the coordinates of the lower left hand corner and upper right hand corner of the clip bounds. These world coordinates are relative to the world coordinate space created by the picture's view. Therefore this clip rectangle is altered to be a subset of the input picture's view's world clip bounds if necessary. This additional clipping is applied to all rendering done in the window associated with the input picture.

PRIMITIVES

Primitives are clipped to both the standard and the supplied clip rectangles if the window of the currently open picture is the same as the window of the picture passed to Gsetclip.

SEGMENTS

Segments are clipped to both the standard and the supplied clip rectangles if the window of the currently open picture is the same as the window of the picture passed to Gsetclip.

WINDOWS

All windows except the window of the picture passed to Gsetclip will ignore the additional clip bounds.

VIEWS

Views are clipped to both the standard and the supplied clip rectangles if the window of the currently open picture is the same as the window of the picture passed to Gsetclip.

EXAMPLES

BUGS

There can be only one additional clip bounds at one time.

SEE ALSO

Gdraw, Ghide, Gsetclip, Gunsetclip Gsetdeviceclip, Gunsetdeviceclip

DEFINITION

Causes all draws to ignore the clip bounds set by the function 'Gsetclip'.

OPERATION

```
void Gunsetclip()
```

EXAMPLES

BUGS

To reenable the clip bounds again, Gsetclip must be recalled with the original parameters.

SEE ALSO

Gsetclip, Gsetdeviceclip, Gunsetdeviceclip, Gdraw, Ghide

DEFINITION

Specifies a clipping rectangle to be used in addition to the world bounds of the view in the currently open picture. This additional clipping criteria is in effect until the associated function 'Gunsetdeviceclip' is called.

OPERATION

```
void Gsetdeviceclip(dxmin, dymin, dxmax, dymax)
GT_DCOORD dxmin, dymin, dxmax, dymax;
```

The parameters (dxmin, dymin, dxmax, dymax) are the coordinates of the lower left hand corner and upper right hand corner of the clip bounds. These device coordinates are relative to the device coordinate space of any open picture's device. This additional clipping is applied to all rendering done on any devices.

PRIMITIVES

Primitives are clipped to both the standard and the supplied clip rectangles.

SEGMENTS

Segments are clipped to both the standard and the supplied

clip rectangles.

WINDOWS

Windows are clipped to both the standard and the supplied clip rectangles. Note that window borders may not be clipped to the additional clip bounds supplied by Gsetdeviceclip.

VIEWS

Views are clipped to both the standard and the supplied clip rectangles.

EXAMPLES

BUGS

There can be only one additional clip bounds at one time.

SEE ALSO

Gdraw, Ghide, Gsetclip, Gunsetclip Gunsetdeviceclip

Gunsetdeviceclip PGL DRAW OPERATIONS Gunsetdeviceclip

DEFINITION

Causes all draws to ignore the clip bounds set by the function 'Gsetdeviceclip'.

OPERATION

void Gunsetdeviceclip()

EXAMPLES

BUGS

To reenable the clip bounds again, Gsetdeviceclip must be recalled with the original parameters.

SEE ALSO

Gsetdeviceclip, Gsetclip, Gunsetclip, Gdraw, Ghide

Ggettype PGL OPERATIONS Ggettype

DEFINITION

Returns the graphic type of the given graphic object. The type is used to identify what kind of object the graphic object actually is.

OPERATION

GT_TYPE Ggettype(obj)
GT_OBJECT obj;

PRIMITIVES

The primitive's graphic type is returned.

SEGMENTS

The segment's graphic type (GO_SEGMENT or GO_CSEGMENT) is returned.

WINDOWS

The window's graphic type (GO_WINDOW) is returned.

VIEWS

The view's graphic type (GO_VIEW) is returned.

PICTURES

The picture's graphic type (GO_PICTURE) is returned.

SYMBOLIC PICTURES

The symbolic picture's graphic type (GO_SYMBOLICPICTURE) is returned.

DEVICES

The device's graphic type (GO_DEVICE) is returned.

EXAMPLES

BUGS

SEE ALSO

Ggetextrema PGL OPERATIONS Ggetextrema

DEFINITION

Returns the extrema of the given graphic object. The extrema is defined as the coordinates of the lower left hand corner and upper right hand corner of the smallest enclosing orthogonal rectangle enclosing a

graphics object.

OPERATION

```
void Ggetextrema(obj, bounds)
GT_OBJECT obj;
GT_EXTREMA *bounds;/* The GT_EXTREMA values are filled in*/
```

PRIMITIVES

The primitive's extrema is returned in world coordinates.

SEGMENTS

The segment's extrema, which is the smallest orthogonal rectangle enclosing all of the primitives in the segment, is returned in world coordinates.

WINDOWS

The window's extrema is returned in world coordinates as the maximum extrema possible.

VIEWS

The view's graphic type (GO_VIEW) is returned.

PICTURES

The picture's extrema in it's associated device coordinates is returned.

SYMBOLIC PICTURES

Extrema is undefined for this object.

DEVICES

The device's extrema is returned in device coordinates. This may be used to determine the physical dimensions (resolution) of a video device.

EXAMPLES

BUGS

SEE ALSO

DEFINITION

Establishes the given graphic object as the open object of it's type. This is used to speed up and simplify operations that might otherwise need to have extra parameters.

OPERATION

```
int Gopen(obj)
GT_OBJECT obj;
```

A non-zero value is returned if the open operation is undefined for the input graphic object 'obj'.

PRIMITIVES

The open operation is undefined for primitives.

SEGMENTS

The input segment becomes the open segment.

WINDOWS

The input window becomes the open window.

VIEWS

The input view becomes the open view.

PICTURES

The input picture becomes the open picture.

SYMBOLIC PICTURES

The open operation is undefined for symbolic pictures.

DEVICES

The input device becomes the open device.

EXAMPLES

BUGS

SEE ALSO

Gpushopen, Gpopopen, Ggetopen

DEFINITION

This function is for completeness only and does not do anything at this time.

OPERATION

```
int Gclose(obj)
GT_OBJECT obj;
```

PRIMITIVES

SEGMENTS

WINDOWS

VIEWS

PICTURES

SYMBOLIC PICTURES

DEVICES

EXAMPLES

BUGS

SEE ALSO

Gopen, Gpushopen, Gpopopen, Ggetopen

DEFINITION

Returns the currently open graphic object of the given type.

OPERATION

```
GT_OBJECT Ggetopen(type)
GT_TYPE type;
```

A NULL is returned if the open operation is undefined for the input graphic type 'type'.

PRIMITIVES

The open operation is undefined for primitives. NULL is returned.

SEGMENTS

The currently open segment is returned.

WINDOWS

The currently open window is returned.

VIEWS

The currently open view is returned.

PICTURES

The currently open picture is returned.

SYMBOLIC PICTURES

The open operation is undefined for symbolic pictures. NULL is returned.

DEVICES

The currently open device is returned.

EXAMPLES

BUGS

SEE ALSO

Gopen, Gclose, Gpushopen, Gpopopen

DEFINITION

Saves the currently open object of the same type as the given object on an internal stack and then establishes the given graphic object as the open object of it's type. This allows routines to open a graphic object without having to get, save and restore the previously open graphics state.

OPERATION

```
int Gpushopen(obj)
GT_OBJECT obj;
```

A non-zero value (2) is returned if the open operation is undefined for the input graphic object 'obj'. A non-zero value (1) is returned if the internal stack is full. Otherwise zero is returned.

PRIMITIVES

The open operation is undefined for primitives.

SEGMENTS

The input segment becomes the open segment.

WINDOWS

The input window becomes the open window.

VIEWS

The input view becomes the open view.

PICTURES

The input picture becomes the open picture.

SYMBOLIC PICTURES

The open operation is undefined for symbolic pictures.

DEVICES

The input device becomes the open device.

EXAMPLES

BUGS

There is a finite number of objects that can be pushed. This should be configurable by the application.

SEE ALSO

Gopen, Gclose, Gpopopen, Ggetopen

DEFINITION

Retrieves (pops) the object of the same type as the given object from an internal stack and then establishes it as the open object of it's type.

OPERATION

```
int Gpopopen(obj)
GT_OBJECT obj;
```

A non-zero value (2) is returned if the open operation is undefined for the input graphic object 'obj'. A non-zero value (1) is returned if the internal stack is full. Otherwise zero is returned.

PRIMITIVES

The open operation is undefined for primitives.

SEGMENTS

The segment open at the time of the previous call to Gpushopen that had a segment object as the graphic object parameter becomes the open segment.

WINDOWS

The window open at the time of the previous call to Gpushopen that had a window object as the graphic object parameter becomes the open window.

VIEWS

The view open at the time of the previous call to Gpushopen that had a view object as the graphic object parameter becomes the open view.

PICTURES

The picture open at the time of the previous call to Gpushopen that had a picture object as the graphic object parameter becomes the open picture.

SYMBOLIC PICTURES

The open operation is undefined for symbolic pictures.

DEVICES

The device open at the time of the previous call to Gpushopen that had a device object as the graphic object parameter becomes the open device.

EXAMPLES

BUGS

SEE ALSO

Gopen, Gclose, Gpushopen, Ggetopen

Ggetarray PGL OPERATIONS Ggetarray

DEFINITION

Fills out a structure pointed to by 'array' that represents enough information about the given object (except the graphic context

of the object) to generate a copy of it. This can be used by a file system, to automate creation of large quantities of graphics objects, or to make a copy of the given object after it has been deleted.

OPERATION

```
int Ggetarray(obj, array)
GT_OBJECT obj;
GT_ARRAY array;   /* RETURNED  with caller allocated
                             array filled out */
```

Fills out the array with the appropriate values. See the file 'garray.h' for the definition of the various array structures. Returns size of (number of bytes in) the array.

PRIMITIVES

The array is generated reflecting the primitive.

SEGMENTS

The array is generated reflecting the segment.

WINDOWS

The array is generated reflecting the window.

VIEWS

The array is generated reflecting the view.

PICTURES

The array is generated reflecting the picture.

SYMBOLIC PICTURES

The Ggetarray operation is undefined for symbolic pictures.

DEVICES

The array is generated reflecting the device.

EXAMPLES

BUGS

SEE ALSO

Gexecarray, Gapplyarray

DEFINITION

Creates and returns a handle to a copy of the object that the array is gotten from.

PURPOSE

Takes the array that is filled out by a previous Ggetarray function call on a graphics object and creates a image of the graphics object.

OPERATION

```
GT_OBJECT Gexecarray(array)
GT_ARRAY array;
```

EXAMPLES

BUGS

SEE ALSO

Ggetarray, Gapplyarray

DEFINITION

Modifies the given object making it similar to the original object that the array structure was gotten from. This is similar to a series of GinqXXXXX(original object) and GmodXXXXX(object) calls.

OPERATION

```
GT_OBJECT Gapplyarray(object, array)
GT_OBJECT object;
GT_ARRAY array;
```

PRIMITIVES

This function deletes the given object and returns a handle to a new object matching the given array. The new object is in the same position in it's parent segment.

SEGMENTS

The segment is modified reflecting the array.

WINDOWS

The window is modified reflecting the array.

VIEWS

The view is modified reflecting the array.

PICTURES

This function deletes the given object and returns a handle to a new object matching the given array. The new object is in the same position in it's picture list.

SYMBOLIC PICTURES

The Gapplyarray operation is undefined for symbolic pictures.

DEVICES

The device is modified reflecting the array.

EXAMPLES

BUGS

Modifying a picture (and symbolic picture?) may not restore the picture correctly in the view, window, device lists. This will be repaired someday.

SEE ALSO

Ggetarray, Gexecarray

Gnext PGL OBJECT TRAVERSAL OPERATIONS Gnext

DEFINITION

Returns the handle of the next object following the given object from one of the internally maintained lists of graphic data. NULL is returned if no objects follow the given one.

OPERATION

```
GT_OBJECT Gnext(obj)
GT_OBJECT obj;
```

PRIMITIVES

This function returns the handle of the next primitive following the given primitive in the given primitive's segment.

SEGMENTS

This function returns the handle of the next primitive following the given segment in the given segment's segment. However, if the segment has no 'parent' segment, then the segment following the given segment is returned.

WINDOWS

If the window is a subwindow, the this function returns the handle of the next subwindow following the given subwindow in the given subwindows's parent window. However, if the window has no 'parent' window, then the window following the given window is returned.

VIEWS

If the view is a subview, the this function returns the handle of the next subview following the given subview in the given subview's parent view. However, if the view has no 'parent' view, then the view following the given view is returned. The view is modified reflecting the array.

PICTURES

This function returns the handle of the next picture following the given picture.

SYMBOLIC PICTURES

This function returns the handle of the next symbolic picture following the given symbolic picture.

DEVICES

This function returns the handle of the next device following the given device.

EXAMPLES

BUGS

SEE ALSO

Gprev, Gfirst, Glast, Gfirstelement, Glastelement, Gparent

DEFINITION

Returns the handle of the previous object preceding the given object from one of the internally maintained lists of graphic data. NULL is returned if no objects precede the given one.

OPERATION

GT_OBJECT Gprev(obj)
GT_OBJECT obj;

PRIMITIVES

This function returns the handle of the previous primitive preceding the given primitive in the given primitive's segment.

SEGMENTS

This function returns the handle of the previous primitive preceding the given segment in the given segment's segment. However, if the segment has no 'parent' segment, then the segment preceding the given segment is returned.

WINDOWS

If the window is a subwindow, the this function returns the handle of the previous subwindow preceding the given subwindow in the given subwindows's parent window. However, if the window has no 'parent' window, then the window preceding the given window is returned.

VIEWS

If the view is a subview, the this function returns the handle of the previous subview preceding the given subview in the given subview's parent view. However, if the view has no 'parent' view, then the view preceding the given view is returned. The view is modified reflecting the array.

PICTURES

This function returns the handle of the previous picture preceding the given picture.

SYMBOLIC PICTURES

This function returns the handle of the previous symbolic picture preceding the given symbolic picture.

DEVICES

This function returns the handle of the previous device preceding the given device.

EXAMPLES

BUGS

SEE ALSO

Gnext, Gfirst, Glast, Gfirstelement, Glastelement, Gparent

Gfirst PGL OBJECT TRAVERSAL OPERATIONS Gfirst

DEFINITION

Returns the handle of the first object in the internally maintained list of graphic data of which the given object is a member.

OPERATION

```
GT_OBJECT Gfirst(obj)
GT_OBJECT obj;
```

PRIMITIVES

This function returns the handle of the first primitive in the segment in which the given primitive resides.

SEGMENTS

This function returns the handle of the first primitive in the segment in which the given segment resides. However, if the segment has no 'parent' segment, then the first segment in the list of segments is returned.

WINDOWS

If the window is a subwindow, the this function returns the handle of the first subwindow in the parent window in which the given subwindow resides. However, if the window has no 'parent' window, then the first window in the list of windows is returned.

VIEWS

If the view is a subview, the this function returns the handle of the first subview in the parent view in which the given subview resides. However, if the view has no 'parent' view, then the first view in the list of views is returned.

PICTURES

This function returns the handle of the first picture in the list of pictures.

SYMBOLIC PICTURES

This function returns the handle of the first symbolic picture

in the list of symbolic pictures.

DEVICES

This function returns the handle of the first device in the list of devices.

EXAMPLES

BUGS

This barfs if an excised object, which resides in no list, is the given object.

SEE ALSO

Gnext, Gprev, Glast, Gfirstelement, Glastelement, Gparent, Gexcise

Glast PGL OBJECT TRAVERSAL OPERATIONS Glast

DEFINITION

Returns the handle of the last object in the internally maintained list of graphic data of which the given object is a member.

OPERATION

```
GT_OBJECT Glast(obj)
GT_OBJECT obj;
```

PRIMITIVES

This function returns the handle of the last primitive in the segment in which the given primitive resides.

SEGMENTS

This function returns the handle of the last primitive in the segment in which the given segment resides. However, if the segment has no 'parent' segment, then the last segment in the list of segments is returned.

WINDOWS

If the window is a subwindow, the this function returns the handle of the last subwindow in the parent window in which the given subwindow resides. However, if the window has no 'parent' window, then the last window in the list of windows is returned.

VIEWS

If the view is a subview, the this function returns the handle of the last subview in the parent view in which the given subview resides. However, if the view has no 'parent' view, then the last view in the list of views is returned.

PICTURES

This function returns the handle of the last picture in the list of pictures.

SYMBOLIC PICTURES

This function returns the handle of the last symbolic picture in the list of symbolic pictures.

DEVICES

This function returns the handle of the last device in the list of devices.

EXAMPLES

BUGS

This barfs if an excised object, which resides in no list, is the given object.

SEE ALSO

Gnext, Gprev, Gfirst, Gfirstelement, Glastelement, Gparent, Gexcise

Gfirstelement PGL OBJECT TRAVERSAL OPERATIONS Gfirstelement

DEFINITION

Returns the handle of the first child object of the given object. If the object is not a parent object or has no children then NULL is returned.

OPERATION

```
GT_OBJECT Gfirstelement(obj)
GT_OBJECT obj;
```

PRIMITIVES

This function returns the handle of the first primitive in the segment in which the given primitive resides.

SEGMENTS

This function returns the handle of the first primitive in the segment in which the given segment resides. However, if the segment has no 'parent' segment, then the first segment in the list of segments is returned.

WINDOWS

If the window is a subwindow, the this function returns the handle of the first subwindow in the parent window in which the given subwindow resides. However, if the window has no 'parent' window, then the first window in the list of windows is returned.

VIEWS

If the view is a subview, the this function returns the handle of the first subview in the parent view in which the given subview resides. However, if the view has no 'parent' view, then the first view in the list of views is returned.

PICTURES

This function returns the handle of the first picture in the list of pictures.

SYMBOLIC PICTURES

This function returns the handle of the first symbolic picture in the list of symbolic pictures.

DEVICES

This function returns the handle of the first device in the list of devices.

EXAMPLES

BUGS

This barfs if an primitive object, which has no elements, is the given object.
This always barfs cause head/tail point to pictures not elements any more.

SEE ALSO

Gfirst, Gnext, Gprev, Glast, Glastelement, Gparent, Gexcise

DEFINITION

Returns the handle of the last child object of the given object. If the object is not a parent object or has no children then NULL is returned.

OPERATION

GT_OBJECT Glastelement(obj)
GT_OBJECT obj;

PRIMITIVES

This function returns the handle of the last primitive in the segment in which the given primitive resides.

SEGMENTS

This function returns the handle of the last primitive in the segment in which the given segment resides. However, if the segment has no 'parent' segment, then the last segment in the list of segments is returned.

WINDOWS

If the window is a subwindow, the this function returns the handle of the last subwindow in the parent window in which the given subwindow resides. However, if the window has no 'parent' window, then the last window in the list of windows is returned.

VIEWS

If the view is a subview, the this function returns the handle of the last subview in the parent view in which the given subview resides. However, if the view has no 'parent' view, then the last view in the list of views is returned.

PICTURES

This function returns the handle of the last picture in the list of pictures.

SYMBOLIC PICTURES

This function returns the handle of the last symbolic picture in the list of symbolic pictures.

DEVICES

This function returns the handle of the last device in the list of devices.

EXAMPLES

BUGS

This barfs if an primitive object, which has no elements, is the given object.
This always barfs cause head/tail point to pictures not elements any more.

SEE ALSO

Gfirst, Gnext, Gprev, Glast, Gfirstelement, Gparent, Gexcise

DEFINITION

Returns the handle of the parent object of the given object. If the object does not have a parent object then NULL is returned.

OPERATION

```
GT_OBJECT Gparent(obj)
GT_OBJECT obj;
```

PRIMITIVES

This function returns the handle of the segment in which the given primitive resides.

SEGMENTS

This function returns the handle of the segment in which the given segment resides. However, if the segment has no 'parent' segment, then NULL is returned.

WINDOWS

If the window is a subwindow, the this function returns the handle of the parent window in which the given subwindow resides. However, if the window has no 'parent' window, then NULL is returned.

VIEWS

If the view is a subview, the this function returns the handle of the parent view in which the given subview resides. However, if the view has no 'parent' view, then NULL is returned.

PICTURES

This function returns NULL.

SYMBOLIC PICTURES

This function returns NULL.

DEVICES

This function returns NULL.

EXAMPLES

BUGS

SEE ALSO

Gfirst, Gnext, Gprev, Glast, Gfirstelement, Glastelement, Gexcise

DEFINITION

Removes the given graphic object from associations from any other graphic objects and any internally maintained lists.

OPERATION

```
void Gexcise(object)
GT_OBJECT object;
```

PRIMITIVES

SEGMENTS

WINDOWS

given subwindow from it's parent window. However, if the window is not a subwindow it is removed from the internal list of windows.

VIEWS

If the view is a subview, then this function removes the given subview from it's parent view. However, if the view is not a subview it is removed from the internal list of views.

PICTURES

The picture is removed from the internal list of pictures.

SYMBOLIC PICTURES

The picture is removed from the internal list of symbolic pictures.

DEVICES

The device is removed from the internal list of devices.

EXAMPLES

BUGS

This function dies if an object already excised if excised again.

SEE ALSO

Gappend, Gprepend, Gcopy, Ginsert, Gdelete

DEFINITION

Inserts the first of the given objects into the list the second given object resides in immediately before the second given object.

OPERATION

```
void Ginsert(obj, obefore)
GT_OBJECT obj;
GT_OBJECT obefore;
```

PRIMITIVES

This function inserts the primitive 'obj' immediately before the primitive 'obefore' in the segment in which the given primitive 'obefore' resides in.

SEGMENTS

If the segment 'obefore' resides in another segment as a primitive then this function inserts the segment 'obj' immediately before the primitive 'obefore' in the segment in which the given primitive 'obefore' resides in. Otherwise, this function inserts the segment 'obj' immediately before the segment 'obefore' in the internal list of segments.

WINDOWS

If window 'obefore' is a subwindow then this function inserts the window 'obj' immediately before the subwindow 'obefore' in the window in which the given subwindow 'obefore' resides in. This makes window 'obj' a subwindow. Otherwise, this function inserts the window 'obj' immediately before the window

'obefore' in the internal list of windows.

VIEWS

If view 'obefore' is a subview then this function inserts the view 'obj' immediately before the subview 'obefore' in the view in which the given subview 'obefore' resides in. This makes view 'obj' a subview. Otherwise, this function inserts the view 'obj' immediately before the view 'obefore' in the internal list of views.

PICTURES

The picture 'obj' is inserted before the picture 'obefore' in the internal list of pictures.

SYMBOLIC PICTURES

The symbolic picture 'obj' is inserted before the symbolic picture 'obefore' in the internal list of symbolic pictures.

DEVICES

The device 'obj' is inserted before the device 'obefore' in the internal list of devices.

EXAMPLES

BUGS

This function assumes the objects are of the same type and so are compatible residing in the same list. The only exception is that segments may be inserted into lists of primitives.
This function dies if the second object 'obefore' is not in any list.

SEE ALSO

Gappend, Gprepend, Gcopy, Gexcise, Gdelete

DEFINITION

Appends the given object onto the end of the list of the parent object's elements (children).

OPERATION

```
void Gappend(obj, parent)
GT_OBJECT obj;
GT_OBJECT parent;
```

PRIMITIVES

This function appends the primitive 'obj' to the end of the list of primitives in the given segment 'parent'.

SEGMENTS

This function appends the segment 'obj' to the end of the list of primitives in the given segment 'parent'.

WINDOWS

This function appends the window 'obj' to the end of the list of subwindows in the given window 'parent'. The window 'obj' thereby becomes a subwindow of the 'parent' window.

VIEWS

This function appends the view 'obj' to the end of the list of subviews in the given view 'parent'. The view 'obj' thereby becomes a subview of the 'parent' view.

PICTURES

This function is undefined for pictures.

SYMBOLIC PICTURES

This function is undefined for pictures.

DEVICES

This function is undefined for pictures.

EXAMPLES

BUGS

This function assumes that the graphic object 'obj' is of the correct type compatible with being a child of the graphic object 'parent'. The only exception is that segments may be appended onto lists of primitive. Otherwise the resulting effects are indefinable.

SEE ALSO

Ginsert, Gprepend, Gcopy, Gexcise, Gdelete

DEFINITION

Prepends the given object onto the start of the list of the parent

object's elements (children). I.E. the given graphic object 'obj' becomes the first child of the parent graphics object 'parent'.

OPERATION

```
void Gprepend(obj, parent)
GT_OBJECT obj;
GT_OBJECT parent;
```

PRIMITIVES

This function prepends the primitive 'obj' to the start of the list of primitives in the given segment 'parent'.

SEGMENTS

This function prepends the segment 'obj' to the start of the list of primitives in the given segment 'parent'. If the given 'parent' parameter is NULL the segment is prepended to the beginning of the internal list of segments.

WINDOWS

This function prepends the window 'obj' to the start of the list of subwindows in the given window 'parent'. The window 'obj' thereby becomes a subwindow of the 'parent' window. If the given 'parent' parameter is NULL the window is prepended to the beginning of the internal list of windows.

VIEWS

This function prepends the view 'obj' to the start of the list of subviews in the given view 'parent'. The view 'obj' thereby becomes a subview of the 'parent' view. If the given 'parent' parameter is NULL the view is prepended to the beginning of the internal list of views.

PICTURES

This function is undefined for pictures.

SYMBOLIC PICTURES

This function is undefined for pictures.

DEVICES

This function is undefined for pictures.

EXAMPLES

BUGS

This function assumes that the graphic object 'obj' is of the correct type compatible with being a child of the graphic object 'parent'. The only exception is that segments may be prepended onto lists of primitive. Otherwise the resulting effects are indefinable.

SEE ALSO

Ginsert, Gappend, Gcopy, Gexcise, Gdelete

DEFINITION

Deletes the given object. This function removes the given object from any references by any other objects and then frees the memory that was associated with it.

OPERATION

```
void Gdelete(obj)
GT_OBJECT obj;
```

PRIMITIVES

This function deletes the primitive 'obj'.

SEGMENTS

This function deletes the segment 'obj' and it's contents (all primitives and segments within it).

WINDOWS

This function deletes the window 'obj' and all pictures that reference it.

VIEWS

This function deletes the view 'obj' and all pictures that reference it.

PICTURES

This function deletes the picture 'obj'.

SYMBOLIC PICTURES

This function deletes the symbolic picture 'obj' and all it's associated pictures.

DEVICES

This function deletes the device 'obj' and all pictures that reference it. This therefore deletes any instantiation of system level windows that had existed on the device.

EXAMPLES

BUGS

There is no way to delete only a segment w/o deleting it's contents.
There is no way to delete a device and all windows and views who are associated with it.
There is no way to delete a window and all views that reference it.

SEE ALSO

Ginsert, Gappend, Gprepend, Gcopy, Gexcise.

DEFINITION

Makes a copy and returns the handle to it (the copy) of the given object. This function makes an exact copy except it removes any references by/to any other objects. (I.E. an exact copy of the excised given object).

OPERATION

```
GT_OBJECT Gcopy(obj)
GT_OBJECT obj;
```

PRIMITIVES

This function returns a handle to a copy of the primitive 'obj'.

SEGMENTS

This function makes a copy of the given segment 'obj' and it's contents (all primitives and segments within it) and returns a handle to it. The copy of the segment has no parents.

WINDOWS

This function makes a copy of the given window 'obj' and returns a handle to it. The copy has all references to subwindows and associated pictures removed. The copy is a top level window, i.e. it has no parents.

VIEWS

This function makes a copy of the given view 'obj' and returns a handle to it. The copy has all references to

subviews and associated pictures removed. The copy is a top level view, i.e. it has no parents.

PICTURES

This function returns a handle to a copy of the picture 'obj'.

SYMBOLIC PICTURES

This function is undefined for symbolic pictures.

DEVICES

This function makes a copy of the given device 'obj' and returns a handle to it. The copy has all references to associated pictures removed.

EXAMPLES

BUGS

There is no way to delete only a segment w/o deleting it's contents.
There is no way to delete a device and all windows and views who are associated with it.
There is no way to delete a window and all views that reference it.

SEE ALSO

Ginsert, Gappend, Gprepend, Gcopy, Gexcise.

DEFINITION

Deletes the given object's contents. This function removes the given object's contents (other objects) from any references by any other objects and then frees the memory that was associated with them.

OPERATION

```
void Gpurge(obj)
GT_OBJECT obj;
```

PRIMITIVES

This function does nothing.

SEGMENTS

This function deletes the primitives that are in the segment.

WINDOWS

This function deletes all pictures associated with the window.

VIEWS

This function deletes all pictures associated with the view.

PICTURES

This function does nothing.

SYMBOLIC PICTURES

This function does nothing.

DEVICES

This function deletes all pictures associated with the device.

EXAMPLES

BUGS

See delete, there needs to be a more general way of addressing the contents/header/both.

SEE ALSO

Gdelete, Ginsert, Gappend, Gprepend, Gcopy, Gexcise.

PGL EVENT MANAGEMENT

Events are handled by passing around GT_EVENT event 'packets'. These packets contain an abundance of information about the event. Events are stored internally in a FIFO (first in-first out) queue. Both polling and blocked IO is
supported.

GT_EVENT packets have a variety of fields that may be accessed by the application. The functions to do this follow on the succeeding pages. Events are represented by an event bit mask. They are both enabled and identified by this mask. The 'validevents' parameter to Gwaitforevent and Gpollforevent and the 'type' field in a GT_EVENT pack are both of this mask type. See gkbm.h for the latest and most complete information.

This event bit mask has the following possible values.

G_L_WENTDOWNThe left mouse button was pressed.
G_M_WENTDOWNThe middle mouse button was pressed.
G_R_WENTDOWNThe right mouse button was pressed.
G_L_WENTDOWNThe left mouse button was released.
G_M_WENTDOWNThe middle mouse button was released.
G_R_WENTDOWNThe right mouse button was released.
G_L_CLICKThe left mouse button was rapidly pressed and released.

G_M_CLICKThe middle mouse button was rapidly pressed and released.
G_R_CLICKThe right mouse button was rapidly pressed and released.
G_L_DBLCLICKThe left mouse button was rapidly clicked twice.
G_M_DBLCLICKThe middle mouse button was rapidly clicked twice.
G_R_DBLCLICKThe right mouse button was rapidly clicked twice.
G_BUTTONEVENTOne of the mouse buttons was pressed or released.
G_BUTTONDOWNEVENTOne of the mouse buttons was pressed.
G_CLICKEVENTOne of the mouse buttons was clicked.
G_DBLCLICKEVENTOne of the mouse buttons was rapidly clicked twice.
G_MOTIONEVENTThe mouse moved.

G_METAKEYEVENTSome keyboard key other than an ASCII key was pressed.
G_ASCIIKEYEVENTAn ASCII keyboard key was pressed.
G_KEYEVENTSome keyboard key was pressed.

G_INVALIDEVENTThis event can never occur.
G_NULLEVENTThis event is returned when nothing happened(i.e. during a poll).
G_BORDEREVENTThis event occurred in a window border.
G_ROOTEVENTThis event occurred outside windows created by the application.
G_GRABSELECTEDEVENTOther events in this bitmask are to be returned no matter where they may occur. Events that do not match the other events in the bitmask are handled normally.
G_GRABEVENT Other events in this bitmask are to be returned no matter where they may occur. Events that do not match the other events in the bitmask are ignored as if they never occurred(i.e. consumed).

G_ALLEVENTS All mouse and keyboard events.

PGL EVENT MANAGEMENT

The functions that manipulate packets in the queue are:

G_EVENT Gwaitforevent(validevents) waits for and returns valid events.

G_EVENT Gpollforevent(validevents) polls for and returns valid events.

Gputevent(event) puts event into event queue.

DEFINITION

Waits for any one of the given valid events to occur and returns it to the caller of the function. Any events that occur in the meantime, including the aforementioned valid event, are inspected to determine the picture they occurred in. This picture is defined to be the picture

the cursor is over at the time of the event. If this picture's associated view graphic object has a event procedure registered with it, any events that occur that match the view's specified valid event mask are sent to the registered procedure. The procedure has the choice to absorb the event.

PURPOSE

This function can be a used simply to get keyboard and mouse events. It can also be used as the central dispatcher for an event driven application. To accomplish this the application registers callback (notify) procedures with all the view's it will be dealing with and then calls Gwaitforevent with a mask matching the terminating event (if any).

OPERATION

```
GT_EVENT Gwaitforevent(validevents)
GT_EVENTMASK validevents;
```

EXAMPLES

BUGS

SEE ALSO

Gpollforevent, Gputevent.

Gpollforevent PGL EVENT MANAGEMENT Gpollforevent

DEFINITION

Polls to see if any one of the given valid events has occurred and returns it to the caller of the function if found. Otherwise G_NULLEVENT is returned. The event itself is left in the internal queue until Gwaitforevent is called.

PURPOSE

This function is used to check what the next event is the queue is (if any).

OPERATION

```
GT_EVENT Gpollforevent(validevents)
GT_EVENTMASK validevents;
```

EXAMPLES

BUGS

SEE ALSO

Gwaitforevent, Gputevent.

Gputevent PGL EVENT MANAGEMENT
Gputevent

DEFINITION

Puts the given event into the queue so that it is the next event to be read/seen by Gwaitforevent/Gpollforevent.

PURPOSE

This function is used to send events through the event handlers from one part of the application to another.

OPERATION

```
int Gputevent(event)
GT_EVENT event;
```

Zero is returned if successful, otherwise the event queue is full.

EXAMPLES

BUGS

SEE ALSO

Gwaitforevent, Gpollforevent.

PGL EVENT MANAGEMENT

Functions are provided to inquire the particular aspects of any event.

```
Gevent_type(event)
Gevent_value(event)
Gevent_bstate(event)
Gevent_shiftstate(event)
Gevent_deltadevx(event)
Gevent_deltadevy(event)
Gevent_devx(event)
Gevent_devy(event)
Gevent_x(event)
Gevent_y(event)
Gevent_view(event)
Gevent_window(event)
Gevent_picture(event)
Gevent_device(event)
Gevent_time(event)
```

Functions are also provided to set the particular aspects of any event.

```
Gevent_settype(event)
Gevent_setvalue(event)
Gevent_setbstate(event)
Gevent_setshiftstate(event)
Gevent_setdeltadevx(event)
Gevent_setdeltadevy(event)
Gevent_setdevx(event)
Gevent_setdevy(event)
Gevent_setx(event)
Gevent_sety(event)
Gevent_setview(event)
Gevent_setwindow(event)
Gevent_setpicture(event)
Gevent_setdevice(event)
Gevent_settime(event)
```


DEFINITION

Returns the type of the given event in the form of a mask. This is the same mask that would be necessary to request this event as a validevent when passed to Gwaitforevent and Gpollforevent.

PURPOSE

To provide a means to determine the type of event that has occurred. This function is sufficient except for the G_MOTIONEVENT (and G_KEYEVENT) events. In these cases the actual key (or the new position of the cursor) must still be determined.

OPERATION

```
GT_EVENTMASK Gevent_type(event)
GT_EVENT event;
```

The possible values returned are:

```
G_L_WENTDOWN
G_M_WENTDOWN
G_R_WENTDOWN
G_L_WENTDOWN
G_M_WENTDOWN
G_R_WENTDOWN
G_L_CLICK
G_M_CLICK
G_R_CLICK
G_L_DBLCLICK
```

G_M_DBLCLICK
G_R_DBLCLICK
G_INVALIDEVENT
G_NULLEVENT
G_BORDEREVENT
G_ROOTEVENT
G_GRABSELECTEDEVENT
G_GRABEVENT
G_MOTIONEVENT
G_METAKEYEVENT
G_ASCIIKEYEVENT
G_KEYEVENT
G_DBLCLICKEVENT
G_CLICKEVENT
G_BUTTONEVENT
G_BUTTONDOWNEVENT

EXAMPLES

BUGS

This should be called Gevent_mask and GT_EVENTTYPE should go away.

SEE ALSO

Gevent_value, Gevent_bstate, Gevent_shiftstate,
Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy,
Gevent_x, Gevent_y, Gevent_view, Gevent_window,
Gevent_picture, Gevent_device, Gevent_time.

Gevent_value PGL EVENT MANAGEMENT
Gevent_value

DEFINITION

Returns the value of the given event which is different from the event type mask for certain events.

PURPOSE

To provide a means to determine the actual type of event that has occurred. This function is therefore used to determine which particular key when a G_KEYEVENT event occurs.

OPERATION

GT_EVENTVALUE Gevent_value(event)
GT_EVENT event;

EXAMPLES

BUGS

This should be called Gevent_key and return a GT_EVENTKEY type.

SEE ALSO

Gevent_type, Gevent_bstate, Gevent_shiftstate,
Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy,
Gevent_x, Gevent_y, Gevent_view, Gevent_window,
Gevent_picture, Gevent_device, Gevent_time.

Gevent_bstate PGL EVENT MANAGEMENT
Gevent_bstate

DEFINITION

Returns a bit mask representing which mouse buttons were pressed down
at the time of the given event.

PURPOSE

To provide a means to determine which mouse buttons went down or
are still down. On some window systems this is a faster way to detect
the release of a button that waiting for the button wentup event.

OPERATION

```
GT_EVENTVALUE Gevent_bstate(event)
GT_EVENT event;
```

Possible values for get button status are:

```
G_R_HELDDOWN
G_M_HELDDOWN
G_L_HELDDOWN
```

EXAMPLES

BUGS

GT_EVENTVALUE is probably too large a word to use for these 3 bits.

SEE ALSO

Gevent_type, Gevent_value, Gevent_shiftstate,
Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy,
Gevent_x, Gevent_y, Gevent_view, Gevent_window,
Gevent_picture, Gevent_device, Gevent_time.

Gevent_shiftstate PGL EVENT MANAGEMENT
Gevent_shiftstate

DEFINITION

Returns a bit mask representing which shift keys were pressed down at the time of the given event.

PURPOSE

To provide a means to determine which shift keys are down which may alter the interpretation of the actual returned event. Note that shift key events themselves are not returned at this time.

OPERATION

```
GT_EVENTVALUE Gevent_shiftstate(event)
GT_EVENT event;
```

Possible values for get shift status are:

EXAMPLES

BUGS

GT_EVENTVALUE is probably too large a word to use for these bits.

*** NOT IMPLEMENTED ***

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy, Gevent_x, Gevent_y, Gevent_view, Gevent_window, Gevent_picture, Gevent_device, Gevent_time.

DEFINITION

Returns the X device coordinate distance the cursor has moved between the given event
and the previous event that was read. This distance is also returned when the cursor is locked and can be used as a method of determining how much the cursor 'would have been moved' were it not locked.

PURPOSE

To provide a means to determine how much the cursor moved in a coordinate system that is constant with respect to the user. I.E. world coordinate cursor deltas vary widely depending on the actual world bounds of the particular view the cursor is in.

OPERATION

```
GT_DCOORD Gevent_deltadevx(event)
```

```
      GT_EVENT event;

EXAMPLES

BUGS

SEE ALSO

      Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate,
      Gevent_deltadevy, Gevent_devx, Gevent_devy, Gevent_x, Gevent_y,
      Gevent_view, Gevent_window, Gevent_picture, Gevent_device, Gevent_time.


Gevent_deltadevy          PGL EVENT MANAGEMENT              Gevent_deltadevy

DEFINITION

      Returns the Y device coordinate distance the cursor has moved between
      the given event
      and the previous event that was read. This distance is also returned
      when the cursor is locked and can be used as a method of determining
      how much the cursor 'would have been moved' were it not locked.

PURPOSE

      To provide a means to determine how much the cursor moved in a
      coordinate system that is constant with respect to the user. I.E.
      world coordinate cursor deltas vary widely depending on the actual
      world bounds of the particular view the cursor is in.

OPERATION

      GT_DCOORD Gevent_deltadevy(event)
      GT_EVENT event;

EXAMPLES

BUGS

SEE ALSO

      Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate,
      Gevent_deltadevx, Gevent_devx, Gevent_devy, Gevent_x, Gevent_y,
      Gevent_view, Gevent_window, Gevent_picture, Gevent_device, Gevent_time.


Gevent_devx               PGL EVENT MANAGEMENT                  Gevent_devx

DEFINITION

      Returns the X device coordinate of the cursor when the given event
      occurred.

PURPOSE
```

To provide a quick and easy method of determining the position of the cursor in device coordinates.

OPERATION

```
GT_DCOORD Gevent_devx(event)
GT_EVENT event;
```

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devy, Gevent_x, Gevent_y, Gevent_view, Gevent_window, Gevent_picture, Gevent_device, Gevent_time.

DEFINITION

Returns the Y device coordinate of the cursor when the given event occurred.

PURPOSE

To provide a quick and easy method of determining the position of the cursor in device coordinates.

OPERATION

```
GT_DCOORD Gevent_devy(event)
GT_EVENT event;
```

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_x, Gevent_y, Gevent_view, Gevent_window, Gevent_picture, Gevent_device, Gevent_time.

DEFINITION

Returns the X world coordinate of the cursor when the given event occurred.

PURPOSE

To provide a quick and easy method of determining the position of the cursor in world coordinates.

OPERATION

```
GT_COORD Gevent_x(event)
GT_EVENT event;
```

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy, Gevent_y, Gevent_view, Gevent_window, Gevent_picture, Gevent_device, Gevent_time.

DEFINITION

Returns the Y world coordinate of the cursor when the given event occurred.

PURPOSE

To provide a quick and easy method of determining the position of the cursor in world coordinates.

OPERATION

```
GT_COORD Gevent_y(event)
GT_EVENT event;
```

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy, Gevent_x, Gevent_view, Gevent_window, Gevent_picture, Gevent_device, Gevent_time.

DEFINITION

Returns the view graphic object the cursor was over when the given event occurred.

PURPOSE

To provide a quick and easy method of determining the view of the event.

OPERATION

```
GT_OBJECT Gevent_view(event)
GT_EVENT event;
```

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy, Gevent_x, Gevent_y, Gevent_window, Gevent_picture, Gevent_device, Gevent_time.

DEFINITION

Returns the window graphic object the cursor was over when the given event occurred.

PURPOSE

To provide a quick and easy method of determining the window of the event.

OPERATION

```
GT_OBJECT Gevent_window(event)
GT_EVENT event;
```

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy,

Gevent_x, Gevent_y, Gevent_view, Gevent_picture, Gevent_device,
Gevent_time.

Gevent_picture PGL EVENT MANAGEMENT
Gevent_picture

DEFINITION

Returns the picture graphic object the cursor was over when the given event
occurred.

PURPOSE

To provide a quick and easy method of determining the picture of the event.

OPERATION

GT_OBJECT Gevent_picture(event)
GT_EVENT event;

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate,
Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy,
Gevent_x, Gevent_y, Gevent_view, Gevent_window, Gevent_device,
Gevent_time.

Gevent_device PGL EVENT MANAGEMENT
Gevent_device

DEFINITION

Returns the device graphic object the cursor was over when the given event
occurred.

PURPOSE

To provide a quick and easy method of determining the device of the event.

OPERATION

GT_OBJECT Gevent_device(event)
GT_EVENT event;

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy, Gevent_x, Gevent_y, Gevent_view, Gevent_window, Gevent_picture, Gevent_time.

DEFINITION

Returns the time the given event occurred.

PURPOSE

To provide a quick and easy method of determining the time of the event.

OPERATION

```
struct gtime *Gevent_time(event)
GT_EVENT event;
```

EXAMPLES

BUGS

SEE ALSO

Gevent_type, Gevent_value, Gevent_bstate, Gevent_shiftstate, Gevent_deltadevx, Gevent_deltadevy, Gevent_devx, Gevent_devy, Gevent_x, Gevent_y, Gevent_view, Gevent_window, Gevent_picture, Gevent_device.

Graphics Context Management

Each object has a graphics context. At this time primitives are the only users of the their context. The context specifies what color, line width, text font etc... to use when the primitive is drawn or picked.

void Gsetopenctx(ctx) sets the currently open context to be context ctx. By default, every primitive inherits the currently open context which determines it's color, etc.

GT_CTX Ggetopenctx() returns currently open context.

int Gpushopenctx(ctx) save currently open context and make ctx the open context.

int Gpopopenctx() restore previously pushed context.

GT_ATT Ggetctxvalue(ctx, which) returns value of context attribute 'which'.

void Gsetopenctxtype(type) set the open context type.

GT_CTXTYPE Ggetopenctxtype() returns the type of the currently open context.

GT_ATT Ggetopenctxvalue(which) returns value of open context attribute 'which'.

void Gsetopenctxvalue(which, value) sets context attribute 'which' to 'value'.

void Gsetopenctxvalues(vararglist) sets context attributes.

GT_CTXTYPE Ggetobjctxtype(obj) returns type of context associated with given object.

GT_CTX Gsetobjctxtype(obj, type) sets type of context to be associated with obj.

void Gsetobjctx(obj, ctx) assigns a context to object obj.

GT_CTX Ggetobjctx(obj) returns context of object obj.

GT_CTX Gsetobjctxvalue(obj, which, value) sets object's attribute 'which' to 'value'.

GT_CTX Gsetobjctxvalues(obj, vararglist) sets object's attributes.

GT_ATT Ggetobjctxvalue(obj, which) returns object's attribute value for 'which'.

GT_CTX Gmakectx(type, va_alist) combines the default ctx for the given type and
the given attribute values and returns the resultant context.

GT_CTX Gderivectx(parentctx, va_alist) combines the parentctx context with the given attribute values are returns the resultant context.

CONTEXT

Every object has a graphics context associated with it. Objects inherit the context called the currently open context when they are created. There is always an open context. There are functions provided to access and modify various aspects of an object's context and the open context. The open context is also associated with the currently open window so that when (and if) the window is reopened, the open context will be set to the context that existed when the window was previously the open window.

The context of an object is a cache of attributes to be associated with the object. At this time all contexts contain information pertaining to color, line width, fill, text height, text spacing, text width, font,

write mode, fill pattern, visibility, edge color. These attributes are all of type GT_ATT the and defaults are indicated below.

The two types of contexts currently supported differ in the number of user attributes they have available. These are application specific and are ignored by the graphics system itself. Context type G_CTXTYPE0 has 16 user defined attributes and G_CTXTYPE1 has only 1.

```
/*****************************************************
      System Context Default defines
*****************************************************/
```

The various attributes are referenced using the following (as the 'which' in the described functions).

GARG_DEFAULTS

GARG_COLOR

GARG_FILL

GARG_FILLCOLOR

Specifies the fill color if GARG_FILL is equal to GARG_SOLID_FILL or if GARG_FILL is equal to GARG_PATTERN_FILL or GARG_ROP_FILL and the pattern supplied has a depth of one(i.e. monochrome).

GARG_PATTERN

There is currently two kinds of bitmaps supported (one and eight bits deep). There is also two methods of rendering bitmaps: filled rectangles and filled-ropped rectangles. Filled rectangles align the pattern on the nearest byte boundary which is an integral multiple of the width of the pattern. This is the fastest method of rendering and is used extensively for background patterns. Filled-ropped rectangles align the pattern at the bottom left edge of the rectangle. This is used to draw cursor patterns and similar 'pictures'.

Specifies the fill pattern if GARG_FILL is equal to GARG_PATTERN_FILL or GARG_ROP_FILL. The supplied pattern is of type GT_BITMAP.

GARG_FILLBACKCOLOR

Specifies the background color of the supplied pattern argument to GARG_PATTERN if it is of depth equal to one(i.e. monochrome).

GARG_LWIDTH

GARG_THEIGHT

GARG_TSPACING

GARG_TWIDTH

GARG_TFONT

GARG_WRITEMODE

GARG_HIDDEN

```
#define GI_DEFAULTCOLOR1
#define GI_DEFAULTTEXTHEIGHT40000000
#define GI_DEFAULTTEXTWIDTH30000000
#define GI_DEFAULTTEXTSPACING40000000
#define GI_DEFAULTFONT0
#define GI_DEFAULTWRITEMODEG_COPYWRITEMODE
#define GI_DEFAULTFILLPATTERN0
#define GI_DEFAULTHIDE0
#define GI_DEFAULTFILLFORECOLOR1
#define GI_DEFAULTFILLBACKCOLOR0
#define GI_DEFAULTNOTIFYPROCNULL
```

Context Type:

The functions provided to access and modify the type of a context are:

```
GT_CTXTYPE Ggetobjctxtype(obj)
GT_OBJECT obj;
```

This function returns the type of the context associated with object 'obj'.

```
void Gsetobjctxtype(obj, type)
GT_OBJECT obj;
GT_CTXTYPE type;
```

This function sets the type of the context associated with object 'obj' to 'type'.

```
GT_CTXTYPE Ggetopenctxtype()
```

This function returns the type of the open context.

```
void Gsetopenctxtype(type)
GT_CTXTYPE type;
```

This function sets the type of the open context to 'type'.

DEFINITION

This function returns the type of the context that is associated with the given object.

PURPOSE

Provides support for application determination of context type and size.

OPERATION

```
GT_CTXTYPE Ggetobjctxtype(obj)
GT_OBJECT obj;
```

EXAMPLES

BUGS

SEE ALSO

Gsetobjctxtype, Ggetopenctxtype, Gsetopenctxtype.

DEFINITION

This function sets the type of the context that is associated with the given object and returns a handle to the given object's new context.

PURPOSE

Provides support for application determination of context type and size.

OPERATION

```
GT_CTX Gsetobjctxtype(obj, type)
GT_OBJECT obj;
GT_CTXTYPE type;
```

EXAMPLES

BUGS

SEE ALSO

Ggetobjctxtype, Ggetopenctxtype, Gsetopenctxtype.

Ggetopenctxtype PGL GRAPHIC CONTEXT SUPPORT
Ggetopenctxtype

DEFINITION

This function returns the type of the open context.

PURPOSE

Provides support for application determination of context type and size.

OPERATION

GT_CTXTYPE Ggetopenctxtype()

EXAMPLES

BUGS

SEE ALSO

Ggetobjctxtype, Gsetobjctxtype, Gsetopenctxtype.

Gsetopenctxtype PGL GRAPHIC CONTEXT SUPPORT
Gsetopenctxtype

DEFINITION

This function sets the type of the open context to be the given type.

PURPOSE

Provides support for application determination of context type and size.

OPERATION

```
void Gsetopenctxtype(type)
GT_CTXTYPE type;
```

EXAMPLES

BUGS

SEE ALSO

Ggetobjctxtype, Gsetobjctxtype, Ggetopenctxtype.

Open Context:

The functions provided to access and modify the open context are:

```
int Gpushopenctx(ctx)
GT_CTX ctx;

      Makes the context 'ctx' the open context, saving the previously
      open context on a stack.

int Gpopopenctx()

      Makes the context that was open before the last call to
      Gpushopenctx the new open context.

GT_CTX Ggetopenctx()

      Returns the open context.

void Gsetopenctx(ctx)
GT_CTX ctx;

      Makes the context 'ctx' the open context.

GT_ATT Ggetopenctxvalue(which)
int which;

      Returns the value of the open context attribute specified
      by 'which'.

void Gsetopenctxvalue(which, value)
int which;
GT_ATT value;

      Modifies the open context attribute specified by 'which' and
      sets it to equal 'value'. If which is set to GARG_DEFAULTS, then
      all the context attributes are set to their default values.

void Gsetopenctxvalues(which, value, which, value, ..., 0)
int which;
GT_ATT value;

      Modifies each open context attribute specified by 'which' and
      sets it to equal 'value'. If which is set to GARG_DEFAULTS, then
      all the context attributes are set to their default values.
```


DEFINITION

Makes the given context the open context, saving the previously open context on an internal stack. A nonzero value is returned on stack overflow error (i.e. the stack is not large enough for all the pushed contexts).

PURPOSE

Provides an easy way to temporarily change the open context.

OPERATION

```
int Gpushopenctx(ctx)
GT_CTX ctx;
```

EXAMPLES

BUGS

There is only a finite amount of internal stack space (room for approximately 20 entries).

SEE ALSO

Gpopopenctx, Ggetopenctx, Gsetopenctx, Ggetopenctxvalue, Gsetopenctxvalue, Gsetopenctxvalues.

DEFINITION

Restores the previously open context that was open before the last call to Gpushopenctx as the new open context. A nonzero value is returned on stack underflow (i.e. there are currently no contexts pushed on the internal stack to pop off).

PURPOSE

Provides an easy way to temporarily change the open context.

OPERATION

```
int Gpopopenctx()
```

EXAMPLES

BUGS

There is only a finite amount of internal stack space (room for approximately 20 entries).

SEE ALSO

Gpushopenctx, Ggetopenctx, Gsetopenctx, Ggetopenctxvalue, Gsetopenctxvalue, Gsetopenctxvalues.

DEFINITION

Returns the handle of the currently open context.

PURPOSE

Provides an method to get the currently open context which will be assigned to all objects that are made while it is open.

OPERATION

GT_CTX Ggetopenctx()

EXAMPLES

BUGS

SEE ALSO

Gpushopenctx, Gpopopenctx, Gsetopenctx, Ggetopenctxvalue, Gsetopenctxvalue, Gsetopenctxvalues.

DEFINITION

Sets the currently open context to the given context.

PURPOSE

To set the open graphic context, which is the context assigned to all graphic objects when they are created, to the given, application specified context. This way the application specifies the color, line width, etc. attribute defaults to objects it will then create. The application then changes the open context if it wants to change one or more of the default attributes which are being assigned to new created objects.

OPERATION

```
void Gsetopenctx(ctx)
GT_CTX ctx;
```

EXAMPLES

BUGS

SEE ALSO

Gsetobjctxvalue, Gpushopenctx, Gpopopenctx, Ggetopenctx, Ggetopenctxvalue, Gsetopenctxvalue, Gsetopenctxvalues.

DEFINITION

Returns the value of the attribute in the open graphic context as specified by the given attribute keyword 'which'.

PURPOSE

To provide a method by which the application may determine the value of an attribute which is being assigned to all graphic objects when they are created.

OPERATION

```
GT_ATT Ggetopenctxvalue(which)
int which;
```

EXAMPLES

BUGS

SEE ALSO

Gsetobjctxvalue, Gpushopenctx, Gpopopenctx, Ggetopenctx, Gsetopenctx, Gsetopenctxvalue, Gsetopenctxvalues.

DEFINITION

Sets the value of the attribute in the open graphic context as specified by the given attribute keyword 'which' , and the given value.

PURPOSE

To provide a method by which the application may set the value of an attribute which is being assigned to all graphic objects when they are created.

OPERATION

```
void Gsetopenctxvalue(which, value)
int which;
GT_ATT value;
```

EXAMPLES

BUGS

SEE ALSO

Gsetobjctxvalue, Gpushopenctx, Gpopopenctx, Ggetopenctx, Gsetopenctx, Ggetopenctxvalue, Gsetopenctxvalues.

Gsetopenctxvalues PGL GRAPHIC CONTEXT SUPPORT Gsetopenctxvalues

DEFINITION

Sets the values of the attributes in the open graphic context as specified by the given attribute keywords and the given values as listed in the variable argument list.

PURPOSE

To provide a method by which the application may set the multiple values of the attributes which are being assigned to all graphic objects when they are created. This function is faster than multiple calls to Gsetopenctxvalue.

OPERATION

```
void Gsetopenctxvalues(which, value, which, value, ..., 0)
int which;
GT_ATT value;
```

EXAMPLES

BUGS

SEE ALSO

Gsetobjctxvalue, Gpushopenctx, Gpopopenctx, Ggetopenctx, Gsetopenctx, Ggetopenctxvalue, Gsetopenctxvalue.

Object Context:

```
GT_CTX Ggetobjctx(obj)
GT_OBJECT obj;
```

Returns the context of the object 'obj'.

```
void Gsetobjctx(obj, ctx)
GT_OBJECT obj;
GT_CTX ctx;
```

Changes the object 'obj's context to be context 'ctx'.

```
GT_ATT Ggetobjctxvalue(obj, which)
GT_OBJECT obj;
int which;
```

Returns the value of the object 'obj's context attribute specified by 'which'.

```
GT_CTX Gsetobjctxvalue(obj, which, value)
GT_OBJECT obj;
int which;
GT_ATT value;
```

Set the value of the object 'obj's context attribute specified by 'which' to equal 'value'. If which is set to GARG_DEFAULTS, then all the context attributes are set to their default values.

```
GT_CTX Gsetobjctxvalues(obj, which, value, which, value, ..., 0)
GT_OBJECT obj;
int which;
GT_ATT value;
```

Set the each value of the object 'obj's context attribute specified by 'which' to equal 'value'. If which is set to GARG_DEFAULTS, then all the context attributes are set to their default values.

DEFINITION

Returns the handle of the graphic context currently associated with the given object.

PURPOSE

To provide a method by which the application may determine the context of a given object.

OPERATION

```
GT_CTX Ggetobjctx(obj)
GT_OBJECT obj;
```

EXAMPLES

BUGS

SEE ALSO

Gsetobjctx, Ggetobjctxvalue, Gsetobjctxvalue, Gsetobjctxvalues.

Gsetobjctx PGL GRAPHIC CONTEXT SUPPORT
Gsetobjctx

DEFINITION

Sets the graphic context of the given object to be the given context.

PURPOSE

To provide a method by which the application may assign the context of a given object.

OPERATION

```
void Gsetobjctx(obj, ctx)
GT_OBJECT obj;
GT_CTX ctx;
```

EXAMPLES

BUGS

SEE ALSO

Ggetobjctx, Ggetobjctxvalue, Gsetobjctxvalue, Gsetobjctxvalues.

Ggetobjctxvalue PGL GRAPHIC CONTEXT SUPPORT
Ggetobjctxvalue

DEFINITION

Returns the value of the attribute specified by the given keyword 'which' in the graphic context of the given object.

PURPOSE

To provide a method by which the application may determine the value of an attribute of a given object. I.E. use this function to determine what color a given object is.

OPERATION

```
GT_ATT Ggetobjctxvalue(obj, which)
GT_OBJECT obj;
int which;
```

EXAMPLES

BUGS

SEE ALSO

Ggetobjctx, Gsetobjctx, Gsetobjctxvalue, Gsetobjctxvalues.

Gsetobjctxvalue PGL GRAPHIC CONTEXT SUPPORT
Gsetobjctxvalue

DEFINITION

Sets the value of the attribute specified by the given keyword 'which' in the graphic context of the given object to the given value.

PURPOSE

To provide a method by which the application may assign the value of an attribute of a given object. I.E. use this function to assign a color to a given object.

OPERATION

```
GT_CTX Gsetobjctxvalue(obj, which, value)
GT_OBJECT obj;
int which;
GT_ATT value;
```

EXAMPLES

BUGS

SEE ALSO

Ggetobjctx, Gsetobjctx, Ggetobjctxvalue, Gsetobjctxvalues.

Gsetobjctxvalues PGL GRAPHIC CONTEXT SUPPORT
Gsetobjctxvalues

DEFINITION

Sets the values of the attributes specified by the given keywords in the graphic context of the given object to the given values.

PURPOSE

To provide a method by which the application may assign the values of multiple attributes of a given object. Using this function to assign multiple values to an object's context is faster than multiple calls to Gsetobjctxvalue.

OPERATION

```
GT_CTX Gsetobjctxvalues(obj, which, value, which, value, ..., 0)
GT_OBJECT obj;
```

```
      int which;
      GT_ATT value;

EXAMPLES

BUGS

SEE ALSO

      Ggetobjctx, Gsetobjctx, Ggetobjctxvalue, Gsetobjctxvalues.



Given a Context:

      GT_ATT Ggetctxvalue(ctx, which)
      GT_CTX ctx;
      int which;

            Returns the value of the context 'ctx's attribute specified
            by 'which'.

      GT_CTX Gmakectx(type, which, value, which, value,...,0)
      GT_CTXTYPE type;
      int which;
      GT_ATT value;

            Returns the handle of the graphic context of the given type
            combining the defaults for the type and the specified attribute
            values.

      GT_CTX Gderivectx(parentctx, which, value, which, value,...,0)
      GT_CTX parentctx;
      int which;
      GT_ATT value;

            Returns the handle of the graphic context of the result of
            combining the parentctx and the specified attribute values.



Ggetctxvalue             PGL GRAPHIC CONTEXT SUPPORT             Ggetctxvalue

DEFINITION

      Returns the value of the attribute specified by the given keyword
      in the given graphic context.

PURPOSE

      To provide a method by which the application may determine the value of
      an attribute of a given context.

OPERATION
```

```
      GT_ATT Ggetctxvalue(ctx, which)
      GT_CTX ctx;
      int which;

EXAMPLES

BUGS

SEE ALSO

      Ggetobjctxvalue.


Gmakectx                PGL GRAPHIC CONTEXT SUPPORT                 Gmakectx

DEFINITION

      Makes a context out of the defaults for the given context type and the
      given attribute values and returns a handle to it.

PURPOSE

      To provide a method by which the application may create graphic contexts
      without touching the open context.

OPERATION

      GT_CTX Gmakectx(type, which, value, which, value,...,0)
      GT_CTXTYPE type;
      int which;
      GT_ATT value;

EXAMPLES

BUGS

SEE ALSO

      Gderivectx.


Gderivectx                PGL GRAPHIC CONTEXT SUPPORT
Gderivectx

DEFINITION

      Makes a context out of the values of the attributes of the given
      context and the given attribute values and returns a handle to it.

PURPOSE
```

To provide a method by which the application may create graphic contexts without touching the open context. GARG_DEFAULTS is a valid keyword and makes this function very similar to Gmakectx except that the context type is determined implicitly from the given context.

OPERATION

```
GT_CTX Gderivectx(parentctx, which, value, which, value,...,0)
GT_CTX parentctx;
int which;
GT_ATT value;
```

EXAMPLES

BUGS

SEE ALSO

Gmakectx.

PGL TAG MANAGEMENT

Similar to the graphical context associated with every object there is an ID or tag value also associated with every object. There is likewise an open tag value that is assigned objects when the are created. There are functions to access and modify the open tag as well as an objects tag value. However the open tag is not associated with the open window and is simply a graphics system state. Note that the taglo and taghi functions assume the tag to be a long word.

The application may use this value to identify an object or to provide specific information about the object (i.e. it could be a pointer to data associated with the object, though the user attributes in the graphic context can also be used for this). The search functions are provided to search rapidly
for objects with specified tag values.

The functions provided follow. A Brief description:

Gsetopentag(tag) sets currently open tag value to tag. By default, all objects receive a tag value equal to the currently open tag value.

GT_TAG Ggetopentag() returns currently open tag value.

Gsettag(obj, tag) set object obj's tag value to equal tag.

GT_TAG Ggettag(obj) returns object obj's tag value.

Gsettaghi(obj, tag) set 'high' half of tag value of object.

Gsettaglo(obj, tag) set 'low' half of tag value of object.

GT_INT16 Ggettaghi(obj) returns 'high' half of tag value of object.

GT_INT16 Ggettaglo(obj) returns 'low' half of tag value of object.

Gsetopentag PGL TAG MANAGEMENT Gsetopentag

DEFINITION

Sets the current open tag to equal the given value.

PURPOSE

To set subsequently created object's tag values to the given value.

OPERATION

```
void Gsetopentag(tag)
GT_TAG tag;
```

EXAMPLES

BUGS

SEE ALSO

Ggetopentag, Gsettag, Ggettag, Gsettaghi, Ggettaghi, Gsettaglo, Ggettaglo.

Ggetopentag PGL TAG MANAGEMENT Ggetopentag

DEFINITION

Returns the value of the current open tag.

PURPOSE

To determine the value of the tag which will be assigned to subsequently created objects.

OPERATION

```
GT_TAG Ggetopentag()
```

EXAMPLES

BUGS

SEE ALSO

Gsetopentag, Gsettag, Ggettag, Gsettaghi, Ggettaghi, Gsettaglo, Ggettaglo.

DEFINITION

Assigns the given value to the tag of the given object.

PURPOSE

To change the value of the tag of an object.

OPERATION

```
void Gsettag(obj, tag)
GT_OBJECT obj;
GT_TAG tag;
```

EXAMPLES

BUGS

SEE ALSO

Gsetopentag, Ggetopentag, Ggettag, Gsettaghi, Ggettaghi, Gsettaglo, Ggettaglo.

DEFINITION

Returns the value of the tag of the given object.

PURPOSE

To inquire the value of the tag of an object.

OPERATION

```
GT_TAG Ggettag(obj)
GT_OBJECT obj;
```

EXAMPLES

BUGS

SEE ALSO

Gsetopentag, Ggetopentag, Gsettag, Gsettaghi, Ggettaghi, Gsettaglo, Ggettaglo.

DEFINITION

Assigns the given value to the high shortword of the tag of the given object.

PURPOSE

To provide an standard method to treat the tag value as two separate values.

OPERATION

```
void Gsettaghi(obj, tag)
GT_OBJECT obj;
GT_INT16 tag;
```

EXAMPLES

BUGS

SEE ALSO

Gsetopentag, Ggetopentag, Gsettag, Ggettag, Ggettaghi, Gsettaglo, Ggettaglo.

DEFINITION

Returns the value of the high shortword of the tag of the given object.

PURPOSE

To provide an standard method to treat the tag value as two separate values.

OPERATION

```
GT_INT16 Ggettaghi(obj)
GT_OBJECT obj;
```

EXAMPLES

BUGS

SEE ALSO

Gsetopentag, Ggetopentag, Gsettag, Ggettag, Gsettaghi, Gsettaglo,

Ggettaglo.

Gsettaglo PGL TAG MANAGEMENT Gsettaglo

DEFINITION

Assigns the given value to the low shortword of the tag of the given object.

PURPOSE

To provide an standard method to treat the tag value as two separate values.

OPERATION

```
void Gsettaglo(obj, tag)
GT_OBJECT obj;
GT_INT16 tag;
```

EXAMPLES

BUGS

SEE ALSO

Gsetopentag, Ggetopentag, Gsettag, Ggettag, Gsettaghi, Ggettaghi, Ggettaglo.

Ggettaglo PGL TAG MANAGEMENT Ggettaglo

DEFINITION

Returns the value of the low shortword of the tag of the given object.

PURPOSE

To provide an standard method to treat the tag value as two separate values.

OPERATION

```
GT_INT16 Ggettaglo(obj)
GT_OBJECT obj;
```

EXAMPLES

BUGS

SEE ALSO

Gsetopentag, Ggetopentag, Gsettag, Ggettag, Gsettaghi, Ggettaghi, Gsettaglo.

PGL DATA SEARCH OPERATIONS

void Gpickstate(pickflags, xmin, ymin, xmax, ymax) set criteria for subsequent picks.

int Gpushpickstate(pickflags, xmin, ymin, xmax, ymax) setup new pick state, saving previous on limited size stack.

int Gpoppickstate() restore previous pick state.

GT_OBJECT Gpick(object) find object matching pick criteria.

int Gpickobj(obj) inquire wether object obj matches pick criteria.

void Gsearchstate(flags, method, scope, action, values, destobj) setup criteria for search and actions.

int Gpushsearchstate(flags, method, scope, action, values, destobj) setup new search state, saving old on limited size stack.

int Gpopsearchstate() restore previous search state.

GT_OBJECT Gsearch(obj) traverse obj, performing action on objects matching search criteria.

int Gsearchobject(obj) inquire wether object obj matches search criteria.

GT_OBJECT Gtraverse(object, proc) traverse object according to search state and execute procedure 'proc' whenever an object is encountered.

GT_OBJECT Ggetnextobjectwithname(type, object, name) find object of given type having given name.

Gpick PGL DATA SEARCH OPERATIONS Gpick

DEFINITION

Searches the specified graphic object(s) for an object overlapping and/or contained in the pick state extrema. Returns the handle of the object satisfying the criteria or returns NULL.

OPERATION

GT_OBJECT Gpick(object)

GT_OBJECT object;

PRIMITIVES

This function searches for primitives that match the dimensional criteria specified in the pick state starting at the primitive following the given primitive 'object'. If G_SEARCHDOWN is specified in the pick state then the search will go down into the segments that are invoked by the call primitive. If G_SEARCHACROSSSEGMENTS is specified in the pick state, then upon reaching the end of the given primitive object's segment, the next segment in the internal list of segments will be searched.

SEGMENTS

(1) If the G_SEARCHDOWN bit is set in the pick state:
This function searches for primitives that match the dimensional criteria specified in the pick state starting at the given segment's first primitive.
If no such primitive is found matching the pick criteria then:

(2) If G_SEARCHACROSSSEGMENTS bit is set in the pick state:
This function searches for segments that match the dimensional criteria specified in the pick state starting at the segment following the given primitive 'object'. If the segment is itself a primitive in another segment the search will first search for primitives in the segment's parent segment starting at the primitive following the given segment. If no such primitive is found matching the pick criteria then the search continues with segments following the parent segment.

(3) Else the handle of the given segment object is returned if it matches the pick state criteria else NULL is returned. I.E. a Gpickobj is preformed on the given segment.

WINDOWS

(1) If the G_SEARCHDOWN bit is set in the pick state:
This function picks the segments in the view graphic objects in the pictures that have been associated with the given window. See segment pick.

(3) Else NULL is returned.

VIEWS

(1) If the G_SEARCHDOWN bit is set in the pick state:
This function picks the segment in the given view graphic object. See segment pick.
If no such primitive is found matching the pick criteria then:

(2) If G_SEARCHACROSSVIEWS bit is set in the pick state:
This function searches for segments that match the dimensional

criteria specified in the pick state starting at the segment of the view following the given view 'object'. If the view is a subview of another view object the search will first examine the subviews and then continue with the view following the parent view of the given view object.

(3) Else the handle of the given view object is returned if it matches the pick state criteria else NULL is returned. I.E. a Gpickobj is preformed on the given view.

PICTURES

This function performs a pick on the view graphic object associated with the given picture.

SYMBOLIC PICTURES

This function performs a pick on the each picture object represented by the given symbolic picture.

DEVICES

(1) If the G_SEARCHDOWN bit is set in the pick state: This function picks the segments in the view graphic objects in the pictures that have been associated with the given device. See segment pick.

(3) Else NULL is returned.

EXAMPLES

BUGS

The flag G_SEARCHACROSSWINDOWS not implemented.
The flag G_CONTAINED is not implemented.

SEE ALSO

Gpickobj, Gpickstate, Gpushpickstate, Gpoppickstate.

DEFINITION

Tests the specified graphic object to see if it is overlapping and/or contained in the pick state extrema and returns TRUE if it is, FALSE if not.

OPERATION

```
int Gpickobj(object)
GT_OBJECT object;
```

PRIMITIVES

This function returns a non-zero value if the primitive extrema intersects the pick state extrema. Otherwise zero is returned.

SEGMENTS

This function returns a non-zero value if the segment extrema intersects the pick state extrema. Otherwise zero is returned.

WINDOWS

This function is undefined for window graphic objects.

VIEWS

This function returns a non-zero value if the view's world area extrema intersects the pick state extrema. Otherwise zero is returned.

PICTURES

This function returns a non-zero value if the picture's device area extrema intersects the pick state extrema. Otherwise zero is returned. Note this compares what is usually world coordinates in the pick state extrema against device coordinates.

SYMBOLIC PICTURES

This function is undefined for symbolic pictures.

DEVICES

This function returns a non-zero value if the device's physical extrema intersects the pick state extrema. Otherwise zero is returned. Note this compares what is usually world coordinates in the pick state extrema against device coordinates.

EXAMPLES

BUGS

The flag G_CONTAINED is not implemented.

SEE ALSO

Gpick, Gpickstate, Gpushpickstate, Gpoppickstate.

DEFINITION

Configures the pick state used by the picking functions Gpick() and Gpickobj().

OPERATION

```
void Gpickstate(pickflags, xmin, ymin, xmax, ymax)
int pickflags;
GT_COORD xmin, ymin, xmax, ymax;
```

int pickflags

The pickflags specify how the pick state extrema is compared with, as well as the method of traversal of, the graphic data which is supplied as parameters to Gpick() and Gpickobj(). The pick criteria refers to the particular conditions that must be satisfied to result in a successful pick.

-Methods for comparing an object extrema against the pick state extrema.

G_INTERSECT The pick criteria is satisfied when a graphic object intersects the pick state extrema.

G_CONTAINED The pick criteria is satisfied when a graphic object is contained in the pick state extrema.

-Methods for traversing the graphic data.

G_SEARCHDOWNIf, while traversing the graphic data, a segment or call primitive is encountered, then the primitives inside the segment or the primitives invoked by the call will be traversed. I.E. the search traverses every branch off a node testing each leaf against the pick criteria.

G_SEARCHACROSSSEGMENTSThe traversal of data will continue with the segments following the current if the current fails the pick criteria. If the pick started at the primitive level and all the primitives in the particular segment failed the pick criteria, then the next segment following their parent will be picked.

G_SEARCHACROSSVIEWS The traversal of data will continue with the views following the current if the current fails the pick criteria.

G_SEARCHACROSSWINDOWS The traversal of data will continue with the windows following the current if the current fails the pick criteria.

GT_COORD xmin, ymin, xmax, ymax;

These world coordinate parameters specify the lower left hand corner and upper right hand corner of the pick state extrema.

EXAMPLES

BUGS

The flag G_SEARCHACROSSWINDOWS not implemented.
The flag G_CONTAINED is not implemented.

SEE ALSO

Gpick, Gpickobj, Gpushpickstate, Gpoppickstate.

Gpushpickstate PGL DATA SEARCH OPERATIONS Gpushpickstate

DEFINITION

Configures the pick state used by the picking functions Gpick() and Gpickobj() while saving the previous pick state on an internal stack.

OPERATION

```
int Gpushpickstate(pickflags, xmin, ymin, xmax, ymax)
int pickflags;
GT_COORD xmin, ymin, xmax, ymax;
```

Returns non-zero value if internal stack has overflowed.

BUGS

The internal stack is limited in size (allows ~10 pushes at this time).

SEE ALSO

Gpick, Gpickobj, Gpickstate, Gpoppickstate.

Gpoppickstate PGL DATA SEARCH OPERATIONS Gpoppickstate

DEFINITION

Configures the pick state used by the picking functions Gpick() and Gpickobj() by restoring the pick state that existed previous to the last Gpushpickstate() function call.

OPERATION

int Gpoppickstate()

Returns non-zero value if internal stack has underflowed.

BUGS

The internal stack is limited in size (allows ~10 pushes at this time).

SEE ALSO

Gpick, Gpickobj, Gpickstate, Gpushpickstate.

DEFINITION

Searches the specified graphic object(s) for an object satisfying the search criteria specified in the search state. Returns the handle of the object satisfying the criteria or returns NULL.

OPERATION

GT_OBJECT Gsearch(object)
GT_OBJECT object;

PRIMITIVES

This function searches for primitives that match the search criteria specified in the pick state starting at the primitive following the given primitive 'object'. If G_SEARCHDOWN is specified in the search state then the search will go down into the segments that are invoked by the call primitive. If G_SEARCHACROSSSEGMENTS is specified in the search state, then upon reaching the end of the given primitive object's segment, the next segment in the internal list of segments will be searched.

SEGMENTS

(1) If the G_SEARCHDOWN bit is set in the search state:
This function searches for primitives that match the dimensional criteria specified in the search state starting at the given segment's first primitive.
If no such primitive is found matching the search criteria then:

(2) If G_SEARCHACROSSSEGMENTS bit is set in the search state:

This function searches for segments that match the dimensional criteria specified in the search state starting at the segment following the given primitive 'object'. If the segment is itself a primitive in another segment the search will first search for primitives in the segment's parent segment starting at the primitive following the given segment. If no such primitive is found matching the search criteria then the search continues with segments following the parent segment.

(3) Else the handle of the given segment object is returned if it matches the search state criteria else NULL is returned. I.E. a Gsearch is preformed on the given segment.

WINDOWS

(1) If the G_SEARCHDOWN bit is set in the search state: This function search the segments in the view graphic objects in the pictures that have been associated with the given window. See segment search.

(3) Else NULL is returned.

VIEWS

(1) If the G_SEARCHDOWN bit is set in the search state: This function search the segment in the given view graphic object. See segment search.
If no such primitive is found matching the search criteria then:

(2) If G_SEARCHACROSSVIEWS bit is set in the search state: This function searches for segments that match the dimensional criteria specified in the search state starting at the segment of the view following the given view 'object'. If the view is a subview of another view object the search will first examine the subviews and then continue with the view following the parent view of the given view object.

(3) Else the handle of the given view object is returned if it matches the search state criteria else NULL is returned. I.E. a Gsearch is preformed on the given view.

PICTURES

This function performs a search on the view graphic object associated with the given picture.

SYMBOLIC PICTURES

This function performs a search on the each picture object represented by the given symbolic picture.

DEVICES

(1) If the G_SEARCHDOWN bit is set in the search state:

This function search the segments in the view graphic objects in the pictures that have been associated with the given device. See segment search.

(3) Else NULL is returned.

EXAMPLES

BUGS

The flag G_SEARCHACROSSWINDOWS not implemented.

SEE ALSO

Gsearchobj, Gsearchstate, Gpushsearchstate, Gpopsearchstate, Gtraverse, Ggetnextobjectwithname.

DEFINITION

Tests the specified graphic object to see if it satisfies the criteria specified in the search state and returns TRUE if it is, FALSE if not.

OPERATION

```
int Gsearchobj(object)
GT_OBJECT object;
```

PRIMITIVES

This function returns a non-zero value if the primitive satisfies the search state criteria. Otherwise zero is returned.

SEGMENTS

This function returns a non-zero value if the segment satisfies the search state criteria. Otherwise zero is returned.

WINDOWS

This function returns a non-zero value if the window satisfies the search state criteria. Otherwise zero is returned.

VIEWS

This function returns a non-zero value if the view satisfies the search state criteria. Otherwise zero is returned.

PICTURES

This function returns a non-zero value if the picture satisfies

the search state criteria. Otherwise zero is returned.

SYMBOLIC PICTURES

This function returns a non-zero value if the symbolic picture satisfies the search state criteria. Otherwise zero is returned.

DEVICES

This function returns a non-zero value if the device satisfies the search state criteria. Otherwise zero is returned.

EXAMPLES

BUGS

SEE ALSO

Gsearch, Gsearchstate, Gpushsearchstate, Gpopsearchstate, Gtraverse, Ggetnextobjectwithname.

DEFINITION

Configures the search state used by the searching functions Gsearch() and Gsearchobj(). The search state consists of the search criteria (referring to the particular conditions that an object must satisfy to result in a TRUE match) and the actions to take when an object satisfies the criteria.

OPERATION

```
Gsearchstate(flags, method, scope, action, values, destobj)
int flags, method, action, scope;
char *values;
GT_OBJECT destobj;
```

int flags

The flags specify the method of traversal of the graphic data which is supplied as parameters to Gsearch().

-Methods for traversing the graphic data.

G_SEARCHDOWNIf, while traversing the graphic data, a segment or call primitive is encountered, then the primitives inside the segment or the primitives invoked by the call will be traversed. I.E. the search traverses every branch off a node testing each leaf against the pick

criteria.

G_SEARCHACROSSSEGMENTSThe traversal of data will continue with the segments following the current if the current fails the pick criteria. If the pick started at the primitive level and all the primitives in the particular segment failed the pick criteria, then the next segment following their parent will be picked.

G_SEARCHACROSSVIEWS The traversal of data will continue with the views following the current if the current fails the pick criteria.

G_SEARCHACROSSWINDOWS The traversal of data will continue with the windows following the current if the current fails the pick criteria.

int method

-Methods for comparing object(s) against the given 'values' parameter of the search state.

G_FINDTAGTreat 'values' as a (GT_TAG *) and compare the object's tag with it. The search criteria is satisfied if the two are equal. I.E.

Ggettag(object) == *(GT_TAG *)values

G_FINDTYPE Treat 'values' as a (GT_TYPE *) and compare the object's type with it. The search criteria is satisfied if the two are equal. I.E.

Ggettype(object) == *(GT_TYPE *)values

G_FINDATTRIBUTE Treat 'values' as a (GT_TYPE *) followed by (GT_ATT)'s and compare the object's particular context attribute(s) with it. The search criteria is satisfied if the two are equal. I.E.

Ggetobjctxvalue(object, *(GT_TYPE *)values) == *(GT_ATT *)((GT_TYPE *)values+1)

so that 'values' is a pointer to 'which' attribute in the context is to be compared followed by one or more desired 'values' of this attribute.

G_FINDCTX Treat 'values' as a (GT_CTX *) and compare the object's type with it. The search criteria is satisfied if the two are equal. I.E.

Ggetobjctx(object) == *(GT_CTX *)values

int action

-Action(s) to execute when a object is found that matches the search criteria.

G_DOCUT Remove the object from it's present list and add it the search state's destobj object. In essence this does a Gexcise(object) then a Gappend(object, destobj).

G_DOCOPYMake a copy of the object and add it to the search state's destobj object. In essence this does a newobj = Gcopy(object) then a Gappend(newobj, destobj).

G_DODELETEDeletes the object.

G_DONOTHINGReturns the handle of the object that satisfies the search criteria to the caller of Gsearch().

int scope

-How to interpret the data that the given 'values' search state parameter points to.

G_ONEVALUEThere is only one value to compare against each candidate graphic object.

G_RANGEVALUESThere is a range of values to compare against each candidate graphic object. I.E. the maximum of the range follows the minimum which is pointed to by the search state 'values' pointer and the candidate object satisfies the criteria if the object value is >= the minimum of the range and <= the maximum of the range.

G_ARRAYVALUESThere are a multitude of values to compare against each candidate graphic object. In this case the scope parameter is a number >= 2 which represents the number of values to compare against each object and the search state 'values' parameter is a pointer to the list of values.

char *values;

-The values to compare against each object during the search. The values are interpreted differently according to the other parameters described above.

GT_OBJECT destobj;

-The destination graphic object to use with some of the predefined (canned) actions.

EXAMPLES

BUGS

The flag G_SEARCHACROSSWINDOWS not implemented.

SEE ALSO

Gsearch, Gsearchobj, Gpushsearchstate, Gpopsearchstate, Gtraverse, Ggetnextobjectwithname.

Gpushsearchstate PGL DATA SEARCH OPERATIONS Gpushsearchstate

DEFINITION

Configures the search state used by the searching functions Gsearch() and Gsearchobj() while saving the previous search state on an internal stack.

OPERATION

```
int Gpushsearchstate(flags, method, scope, action, values, destobj)
int flags, method, action, scope;
char *values;
GT_OBJECT destobj;
```

Returns non-zero value if internal stack has overflowed.

EXAMPLES

BUGS

The internal stack is limited in size (allows ~10 pushes at this time).

SEE ALSO

Gsearch, Gsearchobj, Gsearchstate, Gpopsearchstate, Gtraverse, Ggetnextobjectwithname.

Gpopsearchstate PGL DATA SEARCH OPERATIONS Gpopsearchstate

DEFINITION

Configures the search state used by the searching functions Gsearch() and Gsearchobj() by restoring the search state that existed previous to the last Gpushsearchstate() function call.

OPERATION

```
int Gpopsearchstate()
```

Returns non-zero value if internal stack has underflowed.

EXAMPLES

BUGS

The internal stack is limited in size (allows ~10 pushes at this time).

SEE ALSO

Gsearch, Gsearchobj, Gsearchstate, Gpushsearchstate, Gtraverse, Ggetnextobjectwithname.

DEFINITION

Traverses graphic objects exactly like function Gsearch() and calls the given procedure 'proc' with each graphic object as it is encountered. This allows the application to implement it's own version of Gsearch().

OPERATION

```
GT_OBJECT Gtraverse(object, proc)
GT_OBJECT object;
int (*proc)(GT_OBJECT obj);
```

object

Graphic object at start of objects to traverse.

int (*proc)(GT_OBJECT obj);

Procedure called for every graphic object traversed. If a non-zero value is returned from this procedure then the search is stopped and the current object is returned to the caller of Gtraverse().

EXAMPLES

```
static int draw_selected_objects(obj)
GT_OBJECT obj;
      {
      /* if object matches the search criteria... */
      if (Gsearchobj(obj))
            {
            /* ...draw it */
            Gdraw(obj);
            }
      /* force continuous search */
      return(0);
```

```
            }
      void draw_text(seg)
      GT_OBJECT seg;
            {
            GT_TYPE texttype = GO_TEXT;

            Gpushsearchstate(
            /* flags */
            G_SEARCHDOWN,
            /* method */
            G_FINDTYPE,
            /* scope */
            G_ONEVALUE,
            /* action */
            G_DONOTHING,/* ignored anyway */
            /* values */
            (char *)&texttype,
            /* destobj */
            NULL);            /* ignored anyway */

            Gtraverse(seg, draw_selected_objects);
            Gpopsearchstate();
            }
```

BUGS

SEE ALSO

Gsearch, Gsearchobj, Gsearchstate, Gpushsearchstate, Gpopsearchstate, Ggetnextobjectwithname.

Ggetnextobjectwithname PGL DATA SEARCH OPERATIONS Ggetnextobjectwithname

DEFINITION

Traverses internal list of graphic objects searching for an object of the given type having the given name.

OPERATION

```
GT_OBJECT  Ggetnextobjectwithname(type, object, name)
GT_TYPE type;
GT_OBJECT object;
char *name;
```

type

The type of graphics object desired.

object

Graphic object preceding the first graphic object to

have it's name compared with the given name. If this is NULL, then the search starts at the beginning of the list.

name

The ASCII text name.

PRIMITIVES

This function is undefined for these graphic objects.

SEGMENTS

This function returns the next segment following the given segment with it's name the same as the given name.

WINDOWS

This function returns the next window following the given window with it's name the same as the given name.

VIEWS

This function returns the next view following the given view with it's name the same as the given name.

PICTURES

This function is undefined for these graphic objects.

SYMBOLIC PICTURES

This function is undefined for these graphic objects.

DEVICES

This function returns the next device following the given device with it's name the same as the given name.

BUGS

SEE ALSO

Gsearch, Gsearchobj, Gsearchstate, Gpushsearchstate, Gpopsearchstate.

DEFINITION

Translates the given view graphic object's world coordinate viewpoint.

PURPOSE

To provide an optimized world coordinate translation for view graphic objects. This is faster than a Gmodview() function call. This translation is essentially a pan without the redraw of the graphic data associated with the view.

OPERATION

```
void Gpanview(view, dx, dy)
GT_OBJECT view;
GT_COORD dx, dy;
```

EXAMPLES

BUGS

SEE ALSO

Gvctodc PGL TRANSFORMS SUPPORT Gvctodc

DEFINITION

Converts a viewport coordinate to a device coordinate for the given picture.

PURPOSE

To provide a portable method by which the application may determine actual device coordinates from viewport coordinates. This routine uses the picture's view, window, and device to calculate the desired values.

OPERATION

```
void Gvctodc(picture, vx, vy, dx, dy)
GT_OBJECT picture;
GT_VCOORD vx, vy;
GT_DCOORD *dx, *dy;
```

EXAMPLES

BUGS

SEE ALSO

Gdctovc, Gvtoddelta, Gdtovdelta.

Gdctovc PGL TRANSFORMS SUPPORT Gdctovc

DEFINITION

Converts a device coordinate to a viewport coordinate for the given picture.

PURPOSE

To provide a portable method by which the application may determine viewport coordinates from actual device coordinates. This routine uses the picture's view, window, and device to calculate the desired values.

OPERATION

```
void Gdctovc(picture, dx, dy, vx, vy)
GT_OBJECT picture;
GT_DCOORD dx, dy;
GT_VCOORD *vx, *vy;
```

EXAMPLES

BUGS

SEE ALSO

Gvctodc, Gvtoddelta, Gdtovdelta.

Gvtoddelta PGL TRANSFORMS SUPPORT Gvtoddelta

DEFINITION

Converts a viewport coordinate distance to a device coordinate distance for the given picture.

PURPOSE

To provide a portable method by which the application may determine device coordinate differences from actual viewport coordinate differences. This routine uses the picture's view, window, and device to calculate the desired values.

OPERATION

```
void Gvtoddelta(picture, vx, vy, dx, dy)
GT_OBJECT picture;
GT_VCOORD vx, vy;
GT_DCOORD *dx, *dy;
```

EXAMPLES

BUGS

*** UNIMPLEMENTED ***

SEE ALSO

Gvctodc, Gdctovc, Gdtovdelta.

Gdtovdelta PGL TRANSFORMS SUPPORT Gdtovdelta

DEFINITION

Converts a device coordinate distance to a viewport coordinate distance for the given picture.

PURPOSE

To provide a portable method by which the application may determine viewport coordinate differences from actual device coordinate differences. This routine uses the picture's view, window, and device to calculate the desired values.

OPERATION

```
void Gdtovdelta(picture, dx, dy, vx, vy)
GT_OBJECT picture;
GT_DCOORD dx, dy;
GT_VCOORD *vx, *vy;
```

EXAMPLES

BUGS

SEE ALSO

Gvctodc, Gdctovc, Gvtoddelta.

Gvctowc PGL TRANSFORMS SUPPORT Gvctowc

DEFINITION

Converts a viewport coordinate to a world coordinate for the given view.

PURPOSE

To provide a convenient method by which the application may determine world coordinates from viewport coordinates. This routine uses the given view's viewport and world bounds to calculate the desired values.

OPERATION

```
void Gvctowc(view, vx, vy, wx, wy)
GT_OBJECT view;
GT_VCOORD vx, vy;
GT_COORD *wx, *wy;
```

EXAMPLES

BUGS

SEE ALSO

Gwctovc, Gvtowdelta, Gwtovdelta.

Gwctovc PGL TRANSFORMS SUPPORT Gwctovc

DEFINITION

Converts a world coordinate to a viewport coordinate for the given view.

PURPOSE

To provide a convenient method by which the application may determine viewport coordinates from world coordinates. This routine uses the given view's viewport and world bounds to calculate the desired values.

OPERATION

```
void Gwctovc(view, wx, wy, vx, vy)
GT_OBJECT view;
GT_COORD wx, wy;
GT_VCOORD *vx, *vy;
```

EXAMPLES

BUGS

SEE ALSO

Gvctowc, Gvtowdelta, Gwtovdelta.

Gvtowdelta PGL TRANSFORMS SUPPORT Gvtowdelta

DEFINITION

Converts a viewport coordinate distance to a world coordinate distance for the given view.

PURPOSE

To provide a convenient method by which the application may determine world coordinate distances from viewport coordinate distances. This routine uses the given view's viewport and world bounds to calculate the desired values.

OPERATION

```
      void Gvtowdelta(view, vx, vy, wx, wy)
      GT_OBJECT view;
      GT_VCOORD vx, vy;
      GT_COORD *wx, *wy;

EXAMPLES

BUGS

SEE ALSO

      Gvctowc, Gwctovc, Gwtovdelta.


Gwtovdelta                  PGL TRANSFORMS SUPPORT
Gwtovdelta

DEFINITION

      Converts a world coordinate distance to a viewport coordinate distance
      for the given view.

PURPOSE

      To provide a convenient method by which the application may determine
      viewport coordinate distances from world coordinate distances. This
      routine uses the given view's viewport and world bounds to calculate
      the desired values.

OPERATION

      void Gwtovdelta(view, wx, wy, vx, vy)
      GT_OBJECT view;
      GT_COORD wx, wy;
      GT_VCOORD *vx, *vy;

EXAMPLES

BUGS

SEE ALSO

      Gvctowc, Gwctovc, Gvtowdelta.


Gdctowc                     PGL TRANSFORMS SUPPORT                     Gdctowc

DEFINITION

      Converts a device coordinate to a world coordinate for the given
      picture.

PURPOSE
```

To provide a convenient method by which the application may determine world coordinates from device coordinates. This routine uses the given picture's view, window and device to calculate the desired values.

OPERATION

```
void Gdctowc(picture, devx, devy, wx, wy)
GT_OBJECT picture;
GT_DCOORD devx, devy;
GT_COORD *wx, *wy;
```

EXAMPLES

BUGS

SEE ALSO

Gwctodc, Gdtowdelta, Gwtoddelta.

Gwctodc PGL TRANSFORMS SUPPORT Gwctodc

DEFINITION

Converts a world coordinate to a device coordinate for the given picture.

PURPOSE

To provide a convenient method by which the application may determine device coordinates from world coordinates. This routine uses the given picture's view, window and device to calculate the desired values.

OPERATION

```
void Gwctodc(picture, wx, wy, dx, dy)
GT_OBJECT picture;
GT_COORD wx, wy;
GT_DCOORD *dx, *dy;
```

EXAMPLES

BUGS

SEE ALSO

Gdctowc, Gdtowdelta, Gwtoddelta.

Gdtowdelta PGL TRANSFORMS SUPPORT Gdtowdelta

DEFINITION

Converts a device coordinate distance to a world coordinate distance for the given picture.

PURPOSE

To provide a convenient method by which the application may determine world coordinate distances from device coordinate distances. This routine uses the given picture's view, window and device to calculate the desired values.

OPERATION

```
void Gdtowdelta(picture, devx, devy, wx, wy)
GT_OBJECT picture;
GT_DCOORD devx, devy;
GT_COORD *wx, *wy;
```

EXAMPLES

BUGS

SEE ALSO

Gdctowc, Gwctodc, Gwtoddelta.

Gwtoddelta PGL TRANSFORMS SUPPORT Gwtoddelta

DEFINITION

Converts a world coordinate distance to a device coordinate distance for the given picture.

PURPOSE

To provide a convenient method by which the application may determine device coordinate distances from world coordinate distances. This routine uses the given picture's view, window and device to calculate the desired values.

OPERATION

```
void Gwtoddelta(picture, wx, wy, dx, dy)
GT_OBJECT picture;
GT_COORD wx, wy;
GT_DCOORD *dx, *dy;
```

EXAMPLES

BUGS

SEE ALSO

Gdctowc, Gwctodc, Gdtowdelta.

Gdctosc PGL TRANSFORMS SUPPORT Gdctosc

DEFINITION

Converts a device coordinate to a screen virtual coordinate for the given device graphic object.

PURPOSE

To provide a convenient method by which the application may determine virtual screen coordinates from device coordinates. This routine uses the given device's actual dimensions to calculate the desired values.

OPERATION

```
void Gdctosc(device, dx, dy, sx, sy)
GT_OBJECT device;
GT_DCOORD dx, dy;
GT_SCOORD *sx, *sy;
```

EXAMPLES

BUGS

SEE ALSO

Gsctodc, Gdtosdelta, Gstoddelta.

Gsctodc PGL TRANSFORMS SUPPORT Gsctodc

DEFINITION

Converts a screen virtual coordinate to a device coordinate for the given device graphic object.

PURPOSE

To provide a convenient method by which the application may determine device coordinates from virtual screen coordinates. This routine uses the given device's actual dimensions to calculate the desired values.

OPERATION

```
void Gsctodc(device, vx, vy, dx, dy)
GT_OBJECT device;
GT_SCOORD vx, vy;
```

```
      GT_DCOORD *dx, *dy;

EXAMPLES

BUGS

SEE ALSO

      Gdctosc, Gdtosdelta, Gstoddelta.


Gdtosdelta                  PGL TRANSFORMS SUPPORT
Gdtosdelta

DEFINITION

      Converts a device coordinate distance to a virtual screen coordinate
      distance for the given device.

PURPOSE

      To provide a convenient method by which the application may determine
      virtual screen coordinate distances from device coordinate distances.
      This routine uses the given device's actual dimensions to calculate
      the desired values.

OPERATION

      void Gdtosdelta(device, devx, devy, sx, sy)
      GT_OBJECT device;
      GT_DCOORD devx, devy;
      GT_SCOORD *sx, *sy;

EXAMPLES

BUGS

      *** UNIMPLEMENTED ***

SEE ALSO

      Gdctosc, Gsctodc, Gstoddelta.


Gstoddelta                  PGL TRANSFORMS SUPPORT
Gstoddelta

DEFINITION

      Converts a virtual screen coordinate distance to a device coordinate
      distance for the given device.

PURPOSE
```

To provide a convenient method by which the application may determine device coordinate distances from virtual screen coordinate distances. This routine uses the given device's actual dimensions to calculate the desired values.

OPERATION

```
void Gstoddelta(device, sx, sy, dx, dy)
GT_OBJECT device;
GT_SCOORD sx, sy;
GT_DCOORD *dx, *dy;
```

EXAMPLES

BUGS

*** UNIMPLEMENTED ***

SEE ALSO

Gdctosc, Gsctodc, Gdtosdelta.

Gwctosc PGL TRANSFORMS SUPPORT Gwctosc

DEFINITION

Converts a world coordinate to a screen virtual coordinate for the given picture graphic object.

PURPOSE

To provide a convenient method by which the application may determine virtual screen coordinates from world coordinates. This routine uses the given picture's view, window and device to calculate the desired values.

OPERATION

```
void Gwctosc(picture, wx, wy, sx, sy)
GT_OBJECT picture;
GT_COORD wx, wy;
GT_SCOORD *sx, *sy;
```

EXAMPLES

BUGS

SEE ALSO

```
Gzoom(win, curxpos, curypos, shift, direction)

Gpan(win, panxmin, panymin, panxmax, panymax)

Gmove(win, cursx, cursy,
            apanxmin, apanymin, apanxmax, apanymax,
            panxmin, panymin, panxmax, panymax, handler)

struct gtile *Grubberbox(bx1, by1, bx2, by2, xmin, ymin, xmax, ymax, corner,
minboxwidth, minboxheight, color, dx, dy)

struct gtile *Gmovebox(x1, y1, x2, y2, xmin, ymin, xmax, ymax, color, dx, dy)
```


DEFINITION

Reduces or increases a view graphic object's world coordinate space.

PURPOSE

To magnify or demagnify the graphics in a view. This function is optimized for this particular process.

OPERATION

```
void Gzoom(view, curxpos, curypos, shift, direction)
GT_OBJECT view;
GT_COORD curxpos, curypos;
int shift;
int direction;
```

The minimum and maximum world size Gzoom() will allow is limited.

view

This is the view graphic object whose world coordinate space will be changed.

curxpos, curypos

This is the position, in world coordinates, around which the zoom occurs. Hueristically this is the cursor position which is over the same displayed graphic detail before and after the zoom, no matter where the cursor in the view. This means that the ratio of the distance from curxpos to the left of the view's world boundary to the distance from curxpos to the right is preserved during the zoom. This is true for the Y ratios also.

shift

This specifies how much the world coordinate space is changed. The greater the given shift parameter is, the less the world space bounds are changed. The actual amount, the same is X and Y directions, is

original_size * (1 + 1/(2^shift)).

i.e. size after zoomin is

original_size - (original_size >> shift)

and size after zoomout is

original_size + (original_size >> shift).

direction

Specifies whether a zoom in (the world space is shrunk, the displayed graphics appears to grow larger) or a zoom out (the world space is enlarged, the displayed graphics appears to shrink).

The given direction can be either:

G_ZOOMIN

G_ZOOMOUT

EXAMPLES

BUGS

SEE ALSO

Gpan, Gmove.

DEFINITION

Translates a view graphic object's world coordinate space.

PURPOSE

To move the apparent view over a group of displayed graphics. The user appears to move the displayed graphical data, some of it disappearing off the edge of the view and some more appearing on the other side. This function is optimized for this particular process.

OPERATION

```
GT_EVENT Gpan(view, panxmin, panymin, panxmax, panymax)
GT_OBJECT view;
```

GT_COORD panxmin, panymin, panxmax, panymax;

When this is called, subsequent mouse motion causes the given view to translate within the supplied boundaries. Any G_BUTTONEVENT or G_KEYEVENT will terminate operation and this function will then return the terminating event to the caller. During the operation the cursor is 'locked' and not allowed to visibly move (so that the cursor will not move out of the window, ending the pan prematurely).

view

This is the view graphic object whose world coordinate space will be changed.

panxmin, panymin, panxmax, panymax

This is the boundary, in world coordinates, within which the panning occurs. Hueristically this is the interesting area, part of which will be displayed at all times. This means that the view minimum world X coordinate is always less than the given panxmax and that the view maximum world X coordinate is always greater than panxmin. Similarly for the Y coordinates.

EXAMPLES

BUGS

SEE ALSO

Gzoom, Gmove.

DEFINITION

Provides interactive support for moving an object with a motion device.

PURPOSE

Provides application with an interactive interface between the user and an application routine that will be called each time the motion device moves. Support is provided for well defined 'autopanning' (autopanning is when the user moves an object up against the side of a view and the view's displayed graphics translate in the opposite direction, allowing the user to position the object at a position that was not displayed when the user started the move).

OPERATION

GT_EVENT Gmove(picture, cursx, cursy,
apanxmin, apanymin, apanxmax, apanymax,
panxmin, panymin, panxmax, panymax, handler)

```
GT_OBJECT picture;
GT_COORD cursx, cursy;
GT_COORD apanxmin, apanymin, apanxmax, apanymax;
GT_COORD panxmin, panymin, panxmax, panymax;
int (*handler)();
```

When this is called, subsequent mouse motion causes the given view to translate within the supplied boundaries. Any event other than G_MOTIONEVENT will terminate operation and this function will then return the terminating event to the caller. During the operation the cursor is 'locked' and not allowed to visibly move (so that the cursor will not move out of the window, ending the pan prematurely).

picture

This picture has the view graphic object whose world coordinate space will be changed when panning and which is used for cursor motion device to world coordinate transformations.

THIS MUST BE THE CURRENTLY OPEN PICTURE.

cursx, cursy

This is the current cursor position in world coordinates (relative to the given picture's view).

apanxmin, apanymin, apanxmax, apanymax

This is the boundary, in world coordinates, which, if and when the cursor moves outside it, autopanning occurs.

panxmin, panymin, panxmax, panymax

This is the boundary, in world coordinates, within which the panning can occur. Hueristically this is the interesting area, part of which will be displayed at all times. This means that the view minimum world X coordinate is always less than the given panxmax and that the view maximum world X coordinate is always greater than panxmin. Similarly for the Y coordinates.

handler

This is a pointer to a function that will be called whenever movement occurs. It provides the application with the information necessary to update the visual display of the object that is being moved.

It is defined as

```
int our_move_handler(dx, dy, autopan)
GT_COORD dx ,dy;
int autopan;
```

dx, dy

These are the change in world coordinates of the mouse since the last time then handler was called.

autopan

This flag is non-zero if autopanning occurred in order that the previous position plus the deltas (dx, dy) remains a visible position.

EXAMPLES

```
      /*
            Variables used by both our_createrectproc and
            our_interactive_create.
      */
      static int firstime;
      static GT_COORD createx;
      static GT_COORD createy;
      static GT_COORD createstartx;
      static GT_COORD createstartx;
      static GT_CTX createctx;

      static int our_createrectproc(dx, dy, panned)
      GT_COORD dx, dy;
      int panned;
            {
            /* if this is the first time here...*/
            if (firstime)
                  {
                  /* ..reset flag. */
                  firstime = 0;
                  }
            else if (!panned)
                  {
                  /* ..else is we didn't pan, undraw old rectangle. */
                  Gdrawrect(createctx, createstartx, createstarty,
                        createx, createy);
                  }
            /* adjust the movable rectangle point */
            createx += dx;
            createy += dy;
            /* update cursor position, since Gmove has it locked */
            Gsetcursor(createx, createy);
            /* draw the new rectangle */
            Gdrawrect(createctx, createstartx, createstarty, createx,
createy);
            }
      /*************************************************************
      * our_interactive_create
      *
      *     Input: event which starts interactive create.
```

```
*           ctx   graphic context of new primitive
*                 NOTE that createrectproc assumes writemode
*                 attribute in createctx is G_XORWRITEMODE so
*                 it can undraw rectangles.
*
*
*************************************************************/
our_interactive_create(event, ctx)
GT_EVENT event;
GT_CTX ctx;
      {
      GT_OBJECT view = Gevent_view(event);
      GT_OBJECT picture = Gevent_picture(event);
      GT_OBJECT seg;
      GT_COORD wxmin, wymin, wxmax, wymax;

      /*
            Let our_createrectproc know that there are no previous
            rectangles to undraw.
      */
      firstime = 1;
      /*
            Let our_createrectproc know the graphic context to draw
            the interactively resizing rectangles with.
      */
      createctx = ctx;

      /*
            Save the current position of the mouse as the fixed
            point of the rectangle and as the  original position
            of the movable point.
      */
      createstartx = Gevent_x(event);
      createstarty = Gevent_y(event);
      createx = createstartx;
      createy = createstarty;

      /*
            Get and use the world bounds of the view as the
            autopan boundary.

      */
      Ginqview(view, GARG_WORLD, &wxmin, &wymin, &wxmax,&wymax,0);

      /*
            We will put the newly created rectangle into the
            top level segment of the view. If the view hasn't
            a segment, create one for it.
      */
      Ginqview(view, GARG_SEGMENT, &seg, 0);
      if (!seg)
            {
            seg = Gsegment(0);
            Gmodview(view, GARG_SEGMENT, seg, 0);
```

```
            }

        /*
            Let Gmove know which picture the mouse movements are
            to be transformed in and also make Gdrawrect in
            our_createrectproc draw to this picture also.
        */
        Gpushopen(picture);

        /*
            Do the interactive rectangle create. This will return
            the event which the user ended the rectangle create
            which is not saved or used here. Autopanning is allowed
            within all world coordinate space here.
        */
        Gmove(picture, createstartx, createstarty,
        /* autopan boundary */
         wxmin, wymin, wxmax, wymax,
        /* pan boundary */
         G_MINWORLD, G_MINWORLD, G_MAXWORLD, G_MAXWORLD,
        our_createrectproc);

        /*
            Restore the previously open picture.
        */
        Gpopopen(picture);

        /*
            Open the segment to put the rectangle into and establish
            the desired graphic context for the rectangle. Create
            the rectangle and restore the previous open segment and
            open context states.
        */
        Gpushopen(seg);
        Gpushopenctx(fed_createctx);
        Grectangle(createstartx, createstarty, createx,createy);
        Gpopopenctx();
        Gpopopen(seg);
        }
```

BUGS

cursx, cursy probably can be read from the cursor internal to Gmove and not supplied by the application.

SEE ALSO

Gzoom, Gpan.

DEFINITION

Reads a font of given name and type and assigns it the given font number. The font is then invoked by setting the font attribute in a text primitives graphic context to this font number. Conversly, all text primitives which reference a font of the given font number will now render with the new text font just loaded.

PURPOSE

To display text in other than the resident bitmap text font.

OPERATION

```
int Greadfont(fontname, fontnum, fonttype)
char *fontname;
int fontnum;
int fonttype;
```

If font is loaded successfully zero is returned. Otherwise either a file I/O error is returned or the following:

G_ERROR_FONTTYPENOTSUPPORTED

fontname

This is a pointer to a text string representing the name of the text font that is to be loaded.

fontnum

This is the number to be assigned to the font after it is loaded to be used to reference the font from the GARG_FONT attribute in the graphic context. If a font has already been loaded and assigned this font number then the previously loaded font will be deleted and the new font loaded in it's place. Font number 0 is reserved for the internal bitmap font and cannot be used here.

fonttype

This is the kind of font that is to be loaded and converted to the internal PGL text font format.

Currently supported inport font types are:

G_FONTTYPE_PGLBINARYPGL's own binary representation.
Current fonts include stroke fonts:

stroke1.fnt Basic stoke text.
stroke2.fntBasic stroke with more details
stroke3.fntBasic outline font
stroke4.fnt
stroke5.fntBasic italics text.
stroke6.fntFancy italics text.

EXAMPLES

```
      static int readallstrokefonts()
            {
            int fontnum = 0;
            char name[13];

            do    {
                  ++fontnum;
                  sprintf(name,"stroke%d.fnt",fontnum);
                  } while ((status
                  = Greadfont(name, fontnum, G_FONTTYPE_PGLBINARY)) == 0);

            /* return # of fonts loaded */
            return(fontnum - 1);
            }
```

BUGS

G_FONTTYPE_PGLBINARY is the only type of font supported.

The font name is used also as the name of the disk file containing the font definition and therefore must be less than or equal to 8 characters long plus three character extension (For DOS only).

SEE ALSO

Gwritefont, Gdeletefont, Gdeletefont, Ginqurefont, Gfont, Gfontchar

Gwritefont PGL FONT OPERATIONS Gwritefont

DEFINITION

Writes the text font of the given font number in the given font type format and assigns it the given font name.

PURPOSE

This will allow conversion of one text font format to another by reading it in in one format and then writing it out in another. Also allows the application to save a font which it has created using the PGL internal font onto disk and in multiple formats.

OPERATION

```
int Gwritefont(fontname, fontnum, fonttype)
char *fontname;
int fontnum;
int fonttype;
```

If font is written successfully zero is returned. Otherwise either a file I/O error is returned or the following:

G_ERROR_FONTNOTLOADEDThere is no font having this
font number currently resident.

G_ERROR_FONTTYPENOTSUPPORTED

fontname

This is a pointer to a text string to represent the name of the text font that is to be written.

fontnum

This is the number of the font which has been read and now is to be written.

fonttype

This is the kind of font that is to be converted to and written form the internal PGL text font format.

Currently supported export font types are:

G_FONTTYPE_PGLBINARYPGL's own binary representation.

EXAMPLES

BUGS

G_FONTTYPE_PGLBINARY is the only type of font supported.

The font name is used also as the name of the disk file containing the font definition and therefore must be less than or equal to 8 characters long plus three character extension (For DOS only).

SEE ALSO

Greadfont, Gdeletefont, Ginqurefont, Gfont, Gfontchar

DEFINITION

Deletes the text font of the given font number.

PURPOSE

This will free up the memory that was taken up by the text font.

OPERATION

int Gdeletefont(fontnum)

```
      int fontnum;

            If font is deleted successfully zero is returned. Otherwise
            a error is returned of the following type:

            G_ERROR_FONTNOTLOADEDThere is no font having this
                                    font number currently resident.

      fontnum

            This is the number of the font which has been read and now is to
            be deleted. Font zero cannot be deleted and therefore the given
            fontnum may not equal zero.

EXAMPLES

BUGS

      Fontnum equal 0 is not checked for and will be bad.

SEE ALSO

      Greadfont, Gwritefont, Ginqurefont, Gfont, Gfontchar


Ginquirefont                PGL FONT OPERATIONS
Ginquirefont

DEFINITION

      Inquires the text font of the given font number.

PURPOSE

      This will free up the memory that was taken up by the text font.

OPERATION

      char *Ginquirefont(fontnum)
      int fontnum;

            If font is inquired successfully a non-zero value is returned.
            Otherwise NULL is returned (probably because there is no font
            of the given font number).

      fontnum

            This is the number of the font which is now to be inquired.

EXAMPLES

BUGS

      A (char *)1 is returned because the font name is not now saved with
```

the font definition.
More information should be returned about the font.

SEE ALSO

Greadfont, Gwritefont, Gdeletefont, Ginqurefont, Gfont, Gfontchar

Gfont PGL FONT OPERATIONS Gfont

DEFINITION

Creates a new font.

PURPOSE

This will create a font ready for the application to define characters for using Gfontchar(). It can then be displayed in place of one of the standard fonts or saved using Gwritefont(), or use any of the standard font manipulation routines.

OPERATION

```
int Gfont(fontnum)
int fontnum;
```

*** UNIMPLEMENTED ***

fontnum

This is the number of the font which is now to be created.

EXAMPLES

BUGS

*** UNIMPLEMENTED ***

SEE ALSO

Greadfont, Gwritefont, Gdeletefont, Ginqurefont, Gfontchar

Gfontchar PGL FONT OPERATIONS
Gfontchar

DEFINITION

Adds or replaces a character defintion in the text font of the given font number.

PURPOSE

This will create a font character ready for display or manipulation using the standard font routines.

OPERATION

```
int Gfontchar(fontnum, ch, object_defintion)
int fontnum;
char ch;
GT_OBJECT object_defintion;
```

*** UNIMPLEMENTED ***

fontnum

This is the number of the font to which the new character defintion is to be added.

EXAMPLES

BUGS

*** UNIMPLEMENTED ***

SEE ALSO

Greadfont, Gwritefont, Gdeletefont, Ginqurefont, Gfontchar

Gfilewrite PGL FILE SYSTEM SUPPORT Gfilewrite

DEFINITION

Writes then given object to the file specified by the filestate.

PURPOSE

To save graphic objects to a device (i.e. disk) for more permanent storage or processing.

OPERATION

```
void Gfilewrite(object)
GT_OBJECT object;
```

EXAMPLES

BUGS

Segments called from calls within segments are written each time in their entirety even though is may be the same segment being called many times.
Windows, devices, pictures, views cannot be written at this time.

SEE ALSO

Gfileread, Gfilestate, Gpushfilestate, Gpopfilestate, Gwritefile, Greadfile.

Gfileread PGL FILE SYSTEM SUPPORT Gfileread

DEFINITION

Reads the open file specified by the filestate.

PURPOSE

To read graphic objects from a device (i.e. disk) for more permanent storage or processing.

OPERATION

void Gfileread()

PRIMITIVES

Primitives are read and put into the currently open segment.

EXAMPLES

BUGS

SEE ALSO

Gfilewrite, Gfilestate, Gpushfilestate, Gpopfilestate, Gwritefile, Greadfile.

Gfilestate PGL FILE SYSTEM SUPPORT Gfilestate

DEFINITION

Configures the file state used by the file functions Gfileread() and Gfilewrite() while saving the previous file state on an internal stack.

PURPOSE

To allow the application to setup the file system for subsequent operations.

OPERATION

```
int Gfilestate(filename, fileflags, writehandler, readhandler)
char *filename;
int fileflags;
int (*writehandler)();
```

```
int (*readhandler)();
```

Zero is returned is this function is successful. Errors include internal file state stack overflow and file open errors. The file specified by the given filename parameter is (re)opened each time the given filename or given fileflags change.

filename

The ASCII text name of the file. Note that operating system limitations should be observed here (if the standard writehandler and readhandler are used which are just shells around fwrite and fread 'C' library functions).

fileflags

The given fileflags parameter specifies the type of file contents, file operations, and file systems.

-The file contents types supported are:

G_FILETYPEASCII

The file contains an ASCII representation of graphic data in the form of 'C' language statements. When these statements are compiled and executed as code or read in with the PGL function Gfileread() the graphic data that was written out with Gfilewrite() will be generated.

If file contents are NOT of type G_FILETYPEASCII then data is stored in big-endian format (where the most significant byte is at the lowest address). This means bytes are swapped on writes and reads on INTEL type microprocessors so that binary files are readable by all platforms.

-The file operations supported are:

G_CREATEFILE

When the file is opened it is created. So that if there was a file by the same name, it is deleted. If in a G_WRITEFILE mode 'C' library fopen parameter is "wb". Else if in a G_READFILE operation mode the 'C' library fopen function parameter is "w+b".

If not in G_CREATEFILE operation mode then if in G_WRITEFILE operation mode then the fopen parameter becomes "ab", if in G_READFILE mode

it becomes "rb".

G_WRITEFILE

Specifies how the file is opened, see G_CREATEFILE above.

G_READFILE

Specifies how the file is opened, see G_CREATEFILE above.

G_FILEWRITEFLUSH

The file is flushed every write operation.

-The file system types supported are:

G_NONSTANDARDFILESYSTEM

The file is not opened by Gfilestate() and must be opened by application. This also allows file output to devices other than the disk device (by setting this flag and modifying the read and write handlers - see below).

writehandler

This is a pointer to the function which is called when data is to be written. It is of the form:

```
int writehandler(buffer, count)
char *buffer;
int count;
```

The count actually written is returned.

buffer

Pointer to the data to be written.

count

Number of bytes to write from the buffer to the device.

G_DEFAULTFILEWRITEHANDLER

If writehandler is equal to this, then the internal write handler is used (a shell around fwrite).

readhandler

This is a pointer to the function which is called when data is to be read. It is of the form:

```
int gd_filestdreadhandler(buffer, count)
char *buffer;
int count;
```

The count actually read is returned.

buffer

Pointer to the place to put the data read.

count

Number of bytes to read from the device to the buffer.

G_DEFAULTFILEREADHANDLER

If readhandler is equal to this, then the internal read handler is used (a shell around fread).

EXAMPLES

BUGS

Need an (*openhandler)() to handle conversion of names for application OS independent names easier.
G_FILEWRITEFLUSH is not implemented.

SEE ALSO

Gfilewrite, Gfileread, Gpushfilestate, Gpopfilestate, Gwritefile, Greadfile.

Gpushfilestate PGL DATA SEARCH OPERATIONS Gpushfilestate

DEFINITION

Configures the file state used by the file functions Gfileread() and Gfilewrite() while saving the previous file state on an internal stack.

PURPOSE

To allow the application to use the file system without having to remember how the file state is currently set up and without having to save the state and then restore it when done (though Gpopfilestate should be called).

OPERATION

```
int Gpushfilestate(filename, fileflags, writehandler, readhandler)
char *filename;
int fileflags;
int (*writehandler)();
int (*readhandler)();
```

Returns non-zero value if internal stack has overflowed or a file open error has occurred.

EXAMPLES

BUGS

The internal stack is limited in size (allows ~10 pushes at this time).

SEE ALSO

Gfilewrite, Gfileread, Gfilestate, Gpopfilestate, Gwritefile, Greadfile.

Gpopfilestate PGL DATA SEARCH OPERATIONS Gpopfilestate

DEFINITION

Configures the file state used by the file functions Gfileread() and Gfilewrite() by restoring the file state that existed previous to the last Gpushfilestate() function call.

PURPOSE

To restore the previous filestate without having to know or remember what it was.

OPERATION

```
int Gpopfilestate()
```

Returns non-zero value if internal stack has underflowed or a file re-open error has occurred.

EXAMPLES

BUGS

The internal stack is limited in size (allows ~10 pops at this time).

SEE ALSO

Gfilewrite, Gfileread, Gfilestate, Gpushfilestate, Gwritefile, Greadfile.

Gwritefile PGL DATA SEARCH OPERATIONS Gwritefile

DEFINITION

To write data by directly accessing the writehandler installed in the current file state.

PURPOSE

To allow the application to supplement the file system writing of data.

OPERATION

```
int (*Gwritefile)(buffer, count)
char *buffer;
unsigned int count;
```

Returns count of bytes actually written.

(See Gfilestate (writehandler) for more information).

EXAMPLES

BUGS

This is a lousy (confusing) name for this function.

SEE ALSO

Gfilewrite, Gfileread, Gfilestate, Gpushfilestate, Gpopfilestate, Greadfile.

DEFINITION

To read data by directly accessing the readhandler installed in the current file state.

PURPOSE

To allow the application to supplement the file system reading of data.

OPERATION

```
int (*Greadfile)(buffer, count)
char *buffer;
unsigned int count;
```

Returns count of bytes actually read.

(See Gfilestate (readhandler) for more information).

EXAMPLES

BUGS

This is a lousy (confusing) name for this function.

SEE ALSO

Gfilewrite, Gfileread, Gfilestate, Gpushfilestate, Gpopfilestate, Gwritefile.

DEFINITION

To write the ASCII 'C' language definition of a graphic object.

PURPOSE

To allow the application to do it's own file output. Note that this differs from Gfilewrite in that the context of the given object is not written out and that call primitives and segments are not traversed in any way.

OPERATION

```
void Gwrite(object)
GT_OBJECT object;
```

(See Gfilestate (writehandler) for more information).

EXAMPLES

```
#define MAXFILEOUTPUTBUFFER80
static char fileoutputbuffer[MAXFILEOUTPUTBUFFER];
static int fileoutbuffercount = 0;

static our_filewritehandler(buffer, count)
char *buffer;
int count;
      {
      /* if the buffer isn't big enough... */
      if (fileoutbuffercount + count >= MAXFILEOUTPUTBUFFER)
            {
            /*..adjust count of bytes to copy so don't overrun it */
            count = MAXFILEOUTPUTBUFFER - 1 - fileoutbuffercount;
            }
      /* append the data in the buffer */
      bcopy(buffer,fileoutputbuffer + fileoutputbuffercount, count);
      /* record number bytes in buffer */
      fileoutbuffercount += count;
      /* append data with a '\0' */
      *(fileoutputbuffer + fileoutputbuffercount) = 0;
      }
/*************************************************************
```

```
      our_dump_object

            Generate a text graphic object containing a ASCII string
            representing the PGL function call in 'C' source code that
            would create a copy the given object and display
            this ASCII string at the given dump_object_x, dump_object_y
            in the currently open picture. The text object is added to
            the currently open segment.
      ************************************************************/
      static int our_dump_object(obj, dump_object_x, dump_object_y)
      GT_OBJECT obj;
      GT_COORD dump_object_x;
      GT_COORD dump_object_y;
            {
            GT_OBJECT textobject;

            fileoutbuffercount = 0;
            Gpushfilestate(
                        /* file name (not needed here) */
                        NULL,
                        /* file types */
                        G_FILETYPEASCII
                        | G_FILEWRITEFLUSH
                        | G_CREATEFILE
                        | G_WRITEFILE
                        | G_NONSTANDARDFILESYSTEM,
                        /* call our local routine for output */
                        our_filewritehandler,
                        /* don't need this */
                        G_DEFAULTFILEREADHANDLER);
            /* generate ASCII 'C' function call equivalent of the object */
            Gwrite(obj);
            /* restore previous file state */
            Gpopfilestate();
            /*
                  Generate a text object for retained graphic display
                  of the text string in the currently open segment
            */
            textobject = Gtext(dump_object_x, dump_object_y,
fileoutputbuffer);
            /* display the text in the currently open picture */
            Gdraw(textobject);

            }
BUGS

      This is a lousy interface. Probably should hook up this and Gfilewrite
      to Gtraversal so all options are supported and in a standard way.

SEE ALSO

      Gfilewrite, Gfileread, Gfilestate, Gpushfilestate, Gpopfilestate,
Gwritefile.
```

PGL CURSOR MANAGEMENT

Gsetcursor(x, y) position cursor in open picture at world coordinate.

Gdrawcursor(x, y) draw cursor at device coordinate.

Gsetcursorimage(gp, color) define shape and color of the cursor.

Gsetcursorbounds(xmin, ymin, xmax, ymax) limit cursor movement.

Gsetdefaultcursorbounds() limit cursor to the default boundary.

Gctlcursor(flag) enable/disable automatic system cursor tracking.

Ghidecursor(id) disable display of cursor in the current open picture.

Gunhidecursor(id) enable display of cursor in the current open picture.

Gsetcursor PGL CURSOR MANAGEMENT
Gsetcursor

DEFINITION

Positions the cursor in the currently open picture at the given world coordinate position.

PURPOSE

To provide a method by which the application can manipulate the cursor.

OPERATION

```
void Gsetcursor(x, y)
GT_COORD x;
GT_COORD y;
```

EXAMPLES

BUGS

Perhaps the picture with which to transform the world coordinates to the device coordinates of the cursor should be a parameter. Forgetting that the currently open picture is used leads to bugs.

SEE ALSO

Gdrawcursor, Gsetcursorbounds, Gsetdefaultcursorbounds, Gsetcursorimage, Gctlcursor, Ghidecursor, Gunhidecursor.

Gdrawcursor PGL CURSOR MANAGEMENT
Gdrawcursor

DEFINITION

Draws the cursor at the given device coordinate position.

PURPOSE

To provide a method by which the application can forcefully draw the cursor to the device. Note that clipping, bounds checking, etc. may or may not work for the application using this function.

OPERATION

```
void Gdrawccursor(x, y)
GT_DCOORD x, y;
```

EXAMPLES

BUGS

SEE ALSO

Gsetcursor, Gsetcursorbounds, Gsetdefaultcursorbounds, Gsetcursorimage, Gctlcursor, Ghidecursor, Gunhidecursor.

Gsetcursorbounds PGL CURSOR MANAGEMENT Gsetcursorbounds

DEFINITION

Confines the cursor to the given world coordinate boundary in the currently open picture.

PURPOSE

To provide a method by which the application can prevent the cursor form leaving a specified area without having to lock the cursor and manually update and confine it.

OPERATION

```
void Gsetcursorbounds(xmin, ymin, xmax, ymax)
GT_COORD xmin;
GT_COORD ymin;
GT_COORD xmax;
GT_COORD ymax;
```

EXAMPLES

BUGS

SEE ALSO

Gsetcursor, Gdrawcursor, Gsetdefaultcursorbounds, Gsetcursorimage, Gctlcursor, Ghidecursor, Gunhidecursor.

Gsetdefaultcursorbounds PGL CURSOR MANAGEMENT Gsetdefaultcursorbounds

DEFINITION

Confines the cursor to the default boundary which consists of the edges of the video screen of the device in the currently open picture.

PURPOSE

To provide a method to undo what the function Gsetcursorbounds does. This is the initial condition of the cursor bounds state.

OPERATION

```
void Gsetdefaultcursorbounds()
```

EXAMPLES

BUGS

SEE ALSO

Gsetcursor, Gdrawcursor, Gsetcursorbounds, Gsetcursorimage, Gctlcursor, Ghidecursor, Gunhidecursor.

Gsetcursorimage PGL CURSOR MANAGEMENT Gsetcursorimage

DEFINITION

Defines the cursor image to be the given image. If the given image is monochrome (depth equals one) then the foreground color is the specified color and the background color is transparent.

PURPOSE

To provide a method to change the appearance of the cursor.

OPERATION

```
void Gsetcursorimage(gp, color)
GT_BITMAP gp;
GT_ATT color;
```

EXAMPLES

BUGS

SEE ALSO

Gbitmap, Gsetcursor, Gdrawcursor, Gsetcursorbounds, Gsetdefaultcursorbounds, Gctlcursor, Ghidecursor, Gunhidecursor.

Gctlcursor PGL CURSOR MANAGEMENT Gctlcursor

DEFINITION

Specifies whether the cursor image is to be moved around in response to the mouse by the application (flag is TRUE) or the graphics system (flag is FALSE, the default).

PURPOSE

To provide a method by which the application can explicitly control the behavior of the cursor. Conversely, the behavior of the cursor can be handled by PGL, relieving the application from the overhead of updating it's position all the time.

OPERATION

```
void Gctlcursor(flag)
int flag;
```

EXAMPLES

BUGS

SEE ALSO

Gsetcursor, Gdrawcursor, Gsetcursorbounds, Gsetdefaultcursorbounds, Gsetcursorimage, Ghidecursor, Gunhidecursor.

Ghidecursor PGL CURSOR MANAGEMENT Ghidecursor

DEFINITION

Specifies that the cursor image is not to be displayed in the open picture. If the cursor is currently visible, it is undrawn. A caller identification is passed as a parameter so that a particular Gunhidecursor call(the one that has the same caller identification) is the only call that will reenable display of the cursor. This allows easy nesting of hide/unhide calls without confusion as to what call hid/unhid the cursor last.

PURPOSE

To provide a method by which the application can explicitly control the behavior of the cursor. This can also be used for optimization of drawing speed by hiding the cursor once for multiple object

draws instead of once for each draw.

OPERATION

```
void Ghidecursor(id)
int id;
```

EXAMPLES

BUGS

SEE ALSO

Gsetcursor, Gdrawcursor, Gsetcursorbounds, Gsetdefaultcursorbounds, Gsetcursorimage, Gctlcursor, Gunhidecursor.

DEFINITION

Reenables display of the cursor image in the open picture after a Ghidecursor call. If the cursor is currently invisible, it is made visible. A caller identification is passed as a parameter so that this call will only enable the cursor display if the Ghidecursor that originally hid the cursor had the same caller identification parameter.

PURPOSE

To provide a method by which the application can undo the Ghidecursor function.

OPERATION

```
void Gunhidecursor(id)
int id;
```

EXAMPLES

BUGS

SEE ALSO

Gsetcursor, Gdrawcursor, Gsetcursorbounds, Gsetdefaultcursorbounds, Gsetcursorimage, Gctlcursor, Ghidecursor.

PGL MEMORY MANAGEMENT

DEFINITION

Returns a pointer to an area of the given size, in bytes, that can be used by the function caller. This function does a Gtermin() and a exit(1) if there is not enough free memory to satisfy request.

PURPOSE

To provide a centralized memory resource allocator. This function also has debug malloc support.

OPERATION

```
char *Gmalloc(size)
int size;
```

EXAMPLES

BUGS

Debug malloc is 'ON' all the time (when DEBUG has been defined during compile) and adds ~40 bytes to each allocation size.

SEE ALSO

Gfree, Gdebugmalloc, Gtestmem.

Gfree PGL MEMORY MANAGEMENT Gfree

DEFINITION

Returns the area of memory at the given address to the pool of free memory.

PURPOSE

To provide a centralized memory resource deallocator. This function also has debug malloc support. If Gmallocdebug has been enabled, this function will check the headers and footers of all blocks of memory currently allocated by Gmalloc to see if any have been written on.

OPERATION

```
void Gfree(block)
char *block;
```

EXAMPLES

BUGS

Debug malloc is 'ON' all the time (when DEBUG has been defined during compile) and adds ~40 bytes to each allocation size.
This function checks each time to see if the block's header or footer, created by Gmalloc, has been written over and prints an error if either

has.

SEE ALSO

Gmalloc, Gdebugmalloc, Gtestmem.

Gdebugmalloc PGL MEMORY MANAGEMENT Gdebugmalloc

DEFINITION

Sets/Resets the debug malloc state. When debug malloc is enabled, every Gfree will check all blocks currently allocated by Gmalloc to see if the area in front or in back of the block has been changed. Errors are printed when they are detected.

PURPOSE

To provide a method to turn on and off software that checks for memory areas that are being written to illegally.

OPERATION

```
void Gdebugmalloc(flag)
int flag;         /* TRUE or FALSE */
```

Also setting the string "GMEMDEBUG" to something in the system environment turns on malloc debug form the command line. I.E.

```
set GMEMDEBUG=1
```

turns it on.

EXAMPLES

BUGS

SEE ALSO

Gmalloc, Gfree, Gtestmem.

Gtestmem PGL MEMORY MANAGEMENT Gtestmem

DEFINITION

Tests all blocks currently allocated by Gmalloc to see if the area in front or in back of the block has been changed. Errors are printed when they are detected.

PURPOSE

To provide a method to check the integrity of memory from within the

application at sensitive and/or suspicious places.

OPERATION

int Gtestmem()

Non-zero is returned if an error is detected.

EXAMPLES

BUGS

SEE ALSO

Gmalloc, Gfree, Gdebugmalloc.

PGL BITMAP MANAGEMENT

Gresizebitmap(gp, npattern, newwid, newht) copy and shrink or enlarge bitmap.

G_BITMAP Gbitmap(pattern, sizex, sizey, depth) create a bitmap object.

Gbitmaprotate(gp, angle) NOT currently implemented.

Gloadbitmap(filename, type, gb) load standard bitmaps into bitmap object.

Gfreebitmappattern(gb) free memory associated with the data in a bitmap object.

Gbitmap PGL BITMAP MANAGEMENT Gbitmap

DEFINITION

Returns a handle to a formalized bitmap definition built from the given pattern, device coordinate height and width, and depth (number of bits per pixel).

PURPOSE

To provide a reusable internal representation of standard bitmap data. The GT_BITMAP thus returned can be used as the value of a GARG_PATTERN attribute in a graphic context, as a background fill pattern for a window, or as the definition of a user defined cursor.

OPERATION

GT_BITMAP Gbitmap(pattern, sizex, sizey, depth)
unsigned char *pattern;
int sizex, sizey;

```
      int depth;

EXAMPLES

BUGS

      This GT_BITMAP that is created uses the callers pattern and therefore
      the caller can not free this memory. In the future an option should be
      provided where the pattern is copied to a private area in PGL memory.

SEE ALSO

      Gresizebitmap, Grotatebitmap, Gloadbitmap, Gfreebitmappattern.


Gresizebitmap                PGL BITMAP MANAGEMENT
Gresizebitmap


DEFINITION

      The given GT_BITMAP is changed from the current height and width to the
      given new device coordinate height and width. The buffer pointed
      to by the given 'npattern' is filled with the new bit pattern. If
      'npattern' is NULL then the old buffer where the old bit pattern is will
      be used (i.e. when shrinking bitmaps, one can pass NULL in without
      having to allocate a new buffer and later free the old. Obviously this
      will not be the most efficient use of memory resources on the part of
      the application).

PURPOSE

      To provide a easy way for applications to resize their bitmap data. This
      is useful if the application wishes to run on many platforms which have
      varying video screen sizes and bitmap representations of graphic data
      is to be used.

OPERATION

      void Gresizebitmap(gb, npattern, newwidth, newheight)
      GT_BITMAP gb;
      unsigned char *npattern;
      GT_DCOORD newwidth, newheight;

EXAMPLES

BUGS

      No bitmap compression algorithm is perfect and this one is no exception,
      important artifacts in the bitmap may be lost (the bitmap is not
      examined and there is no artifact detection).

SEE ALSO
```

Gbitmap, Grotatebitmap, Gloadbitmap, Gfreebitmappattern.

Grotatebitmap PGL BITMAP MANAGEMENT Grotatebitmap

DEFINITION

The given GT_BITMAP is rotated through the given angle.

PURPOSE

To provide a easy way for applications to rotate their bitmap data.

OPERATION

```
void Grotatebitmap(gb, angle)
GT_BITMAP gb;
int angle;
```

EXAMPLES

BUGS

*** NOT IMPLEMENTED ***

SEE ALSO

Gbitmap, Gresizebitmap, Gloadbitmap, Gfreebitmappattern.

Gloadbitmap PGL BITMAP MANAGEMENT Gloadbitmap

DEFINITION

The given file is treated as bitmap pattern data and loaded into the given GT_BITMAP.

PURPOSE

To provide a easy way for applications to load their bitmap data.

OPERATION

```
int Gloadbitmap(filename, type, gb)
char *filename;
int type;
GT_BITMAP *gb;
```

Zero is returned upon success. Non-zero errors include file opening errors and unknown bitmaptype.

Current supported bitmap types are:

G_BITMAPTYPE_SUNIM8

G_BITMAPTYPE_SUNRASTER

EXAMPLES

BUGS

There is not now a Gwritebitmap function.

SEE ALSO

Gbitmap, Gresizebitmap, Grotatebitmap, Gfreebitmappattern.

Gfreebitmappattern PGL BITMAP MANAGEMENT Gfreebitmappattern

DEFINITION

The memory associated with the bitmap data in the given GT_BITMAP is returned to the poll of free memory.

PURPOSE

To provide a way for applications to free their bitmap data. This becomes more useful when the data area has been allocated internal to the function Gbitmap when it creates the GT_BITMAP.

OPERATION

```
void Gfreebitmappattern(gb)
GT_BITMAP gb;
```

EXAMPLES

BUGS

SEE ALSO

Gbitmap, Gresizebitmap, Grotatebitmap, Gloadbitmap.

PGL WINDOW STYLE MANAGEMENT

When window graphic objects are created they are assigned their parent's window style. This style object contains information about how the window and it's subsequent subwindows will look. This includes border width, height and
colors, background patterns and colors, and is a convenient method of controlling multiple windows with multiple looks. The window style is only a

suggestion to the lower level window manager. The actual resultant style of a window may be determined by calling Ginqwindow.

GT_WSTYLE Gwstyle(...) Create a window style object.

Ginqwstyle(wstyle, ...) Inquire a window style object.

Gmodwstyle(wstyle, ...) Modify a window style object.

Gdeletewstyle(wstyle) delete a window style object.

Gapplywstylelist(picture, subwstyle_flag,...) change the picture's window or all subsequent subwindows to the given values.

Gapplywstyle(picture, subwstyle_flag, wstyle) change the picture's window or all subsequent subwindows to the given window style.

void Gsetrootwstyle(wstyle) sets the window style for the root window.

GT_WSTYLE Ggetrootwstyle() returns the window style for the root window.

void Gsetrootsubwstyle(wstyle) sets the default window style for subwindows of the windows in the root window.

GT_WSTYLE Ggetrootsubwstyle() returns the default window style for subwindows of the windows in the root window.

GT_WSTYLE Gcopywstyle(wstyle) makes a copy of the wstyle and returns a handle to it.

Gwstyle PGL WINDOW STYLE MANAGEMENT Gwstyle

DEFINITION

Returns the handle to the window style created from the given values.

PURPOSE

To provide a method of encapsulating window appearances for the given values.

OPERATION

GT_WSTYLE Gwstyle(...)

The window style parameters are specified with a variable argument list of GARG_ keyword, value pairs. An allowable value for all keywords is 'G_IGNORE', casted appropriately, which means that the particular value is to remain unchanged

```
      from what it was previously(what a previous open or applied
      wstyle was).

Allowable keywords:Their allowable values:


GARG_WINDOWBORDERWIDTHGT_DCOORDThickness of border on
                                    left and right sides of
                                    window(in device
                                    coordinates).
                          G_DEFAULTBORDERWIDTH is the default.

GARG_WINDOWBORDERHEIGHTGT_DCOORDThickness of border on
                                    top and bottom sides of
                                    window(in device
                                    coordinates).
                          G_DEFAULTBORDERHEIGHT is the default.

GARG_WINDOWBORDERCOLORGT_ATTThe color of any fore-
                                    ground graphics in the
                                    border.
                          G_DEFAULTBORDERCOLOR is the default.

GARG_WINDOWBORDERBACKCOLORGT_ATTThe background (fill)
                                    color of the border.
                          G_DEFAULTBORDERBACKCOLOR is the default.

GARG_WINDOWCOLORGT_ATT  The color (and fill
                                    color) of the interior
                                    of the window.
                          G_DEFAULTWINDOWCOLOR is the default.

GARG_WINDOWBACKCOLORGT_ATTThe fill background
                                    color of the interior
                                    of the window(background
                                    color of the monochrome
                                    fill pattern the window
                                    may have).
                          G_DEFAULTWINDOWBACKCOLOR is the default.

GARG_WINDOWROOTBITMAPGT_BITMAPThe fill pattern of the
                                    interior of the window.
                          NULL (no pattern) is the default.

*Destroy

      void Gdeletewstyle(wstyle);
      GT_WSTYLE wstyle;

*Inquire

      void Ginqwstyle(wstyle, va_alist)
      GT_WSTYLE wstyle;
      va_dcl
```

Takes the input wstyle object and returns the values requested. Valid 'GARG_' keywords are the same as those for wstyle create.

*Modify

```
void Gmodwstyle(wstyle, va_alist)
GT_WSTYLE wstyle;
va_dcl
```

Takes the input segment graphics object and modifies the values requested. Valid 'GARG_' keywords are the same as those for segment create.

EXAMPLES

BUGS

Perhaps each side of the window should have a separate width associated with it.

SEE ALSO

Gopenwstyle, Gpushopenwstyle, Gpopopenwstyle, Gapplywstyle, Gapplywstylelist.

Gapplywstyle PGL WINDOW STYLE MANAGEMENT Gapplywstyle

DEFINITION

Immediately applies the given window style object to the window of the given picture. If the given picture is NULL, then the wstyle is applied to the root window of the PGL window system if there is one.

PURPOSE

To provide a method of changing a window's appearance after it has been created.

OPERATION

```
void Gapplywstyle(picture, wstyle)
int subwstyle_flag;
GT_OBJECT picture;
GT_WSTYLE wstyle;
```

subwstyle_flagint Whether to apply the style to the picture's window (G_FALSE) or as a default for new subwindows of the picture's window (G_TRUE).

EXAMPLES

BUGS

If picture is NULL, this should be applied to the root of whatever window system is managing the screen.

SEE ALSO

Gwstyle, Gopenwstyle, Gpushopenwstyle, Gpopopenwstyle, Gapplywstylelist.

Gapplywstylelist PGL WINDOW STYLE MANAGEMENT Gapplywstylelist

DEFINITION

Immediately applies the given window style values to the window of the given picture. If the given picture is NULL, then the values are applied to the root window of the PGL window system if there is one.

PURPOSE

To provide a method of changing a window's appearance after it has been created without having to create a wstyle object.

OPERATION

```
void Gapplywstylelist(picture, ...)
int subwstyle_flag;
GT_OBJECT picture;
```

The GARG keyword, value parameters valid for Gwstyle are all valid here also.

subwstyle_flagint Whether to apply the style to the picture's window (G_FALSE) or as a default for new subwindows of the picture's window (G_TRUE).

EXAMPLES

BUGS

SEE ALSO

Gwstyle, Gopenwstyle, Gpushopenwstyle, Gpopopenwstyle, Gapplywstyle.

Gsetrootwstyle PGL WINDOW STYLE MANAGEMENT Gsetrootwstyle

DEFINITION

Sets the given window style to be the window style inherited by all

windows that are created that have no parent (i.e. these windows are not subwindows of any windows in the application. These windows therefore are considered subwindows of the root window).

PURPOSE

To provide a method of changing the default window appearance.

OPERATION

```
void Gapplywstylelist(picture, ...)
int subwstyle_flag;
GT_OBJECT picture;
```

The GARG keyword, value parameters valid for Gwstyle are all valid here also.

subwstyle_flagint Whether to apply the style to the picture's window (G_FALSE) or as a default for new subwindows of the picture's window (G_TRUE).

EXAMPLES

BUGS

SEE ALSO

```
void Gsetrootwstyle(wstyle)
GT_WSTYLE wstyle;
GT_WSTYLE Ggetrootwstyle()
void Gsetrootsubwstyle(wstyle)
GT_WSTYLE wstyle;
GT_WSTYLE Ggetrootsubwstyle()
GT_WSTYLE Gcopywstyle(wstyle)
GT_WSTYLE wstyle;
```

PGL LOW LEVEL INTERFACE

Provision has been made for the addition of other graphical output hardware devices. Associated with a 'device' graphic object, these hardware devices may be switched to and from at runtime. When a device becomes active (i.e. a picture object that is associated with the device is opened) it's associated hardware interface becomes the active interface. This interface is standardized and consists of data and a number of functions that must be supplied.

```
struct gdrawfunctions
      {
      void    (*aline)/* draw arbitrarily sloped line */
            (
            short,  /* x1 - x device coordinate point #1 */
            short,  /* y1 - y device coordinate point #1 */
```

```
      short,  /* x2 - x device coordinate point #2 */
      short   /* y2 - y device coordinate point #2 */
      );
  void    (*putchr)/* put a text character at (x, y) */
      (
      short,/* x - device coordinate of lower left of bitmap char */
      short,/* y - device coordinate of lower left of bitmap char */
      char  /* character - ASCII value of character */
      );
void    (*ellipse)/* draw an ellipse */
      (
      short,/* bit mask-denoting quadrants to be rejected/clipped */
      short,  /* yc - y device coordinate of center of ellipse */
      short,  /* xc - x device coordinate of center of ellipse */
      short,/* a - semimajoraxis in device coordinates */
      short /* b - semiminoraxis in device coordinates */
      );
  void    (*clear)/* clear rectangle to fillcolor */
      (
      short,/* x - device coordinate of lower left of area */
      short,/* y - device coordinate of lower left of area */
      short,/* x - device coordinate of upper right of area */
      short /* y - device coordinate of upper right of area */
      );
void    (*pixel)/* set a pixel to the foreground color */
      (
      short,/* x - device coordinate of pixel */
      short /* y - device coordinate of pixel */
      );
void    (*fline)/* draw horizontal line with current pattern */
                /* If the pattern has a depth of 1(monochrome)
                   then background pixels are set to the
                   fillbackcolor and foreground pixels are set
                   to the fillcolor */

      (
      short,/* x - device coordinate of left side of line */
      short,/* x - device coordinate of right side of line */
      short /* y - device coordinate of line */
      );
  void    (*fclear)/* fill rectangle area with fillpattern */
                /* If the pattern has a depth of 1(monochrome)
                   then background pixels are set to the
                   fillbackcolor and foreground pixels are set
                   to the fillcolor */
      (
      short,/* x - device coordinate of lower left of area */
      short,/* y - device coordinate of lower left of area */
      short,/* x - device coordinate of upper right of area */
      short /* y - device coordinate of upper right of area */
      );
void    (*fpattern)/* install pattern as current fill pattern */
      (
      short,/* sizex - #pixels for width of bitmap */
```

```
            short,/* sizey - #pixels for height of bitmap */
            char */* pattern - address of data bits, if NULL then solid
                    fill (i.e. this then equivalent to clear) */
            );
      void    (*rop)/* display pattern starting at pixel (x, y) */
                  /* The coordinate (x, y) may and often will be outside
                    of the video screen bounds */
            (
            short,/* x - device coordinate of left of pattern */
            short,/* y - device coordinate of bottom of pattern */
            short,/* mask - indicating sides of rectangle to be clipped */
            struct gbitmap *     /* address of structure defining pattern */
            );
      void  (*setwritemode)
            (
            GT_ATT/* what logical operation to use to write to pixels */
            );
      void  (*setcolor)
            (
            GT_ATT,/* foreground color */
            GT_ATT, /* fill color */
            GT_ATT/* fill background color */
            );
      void  (*setclipbounds)
            (
            short,/* x - device coordinate of left of bounds */
            short,/* y - device coordinate of bottom of bounds */
            short,/* x - device coordinate of right of bounds */
            short /* y - device coordinate of top of bounds */
            );
      };
```


DEFINITION

Registers and installs new video hardware device and returns the handle to it.

PURPOSE

To provide a method by which alternate output devices can be installed in place of the current one.

OPERATION

```
GT_DRIVER Gvideodriver(buffer, xmin, ymin, xmax, ymax, depth, numcolors,
drawfunctions, init, termin, open, close, videomode, driverdata)
      unsigned char *buffer;
      short xmin, ymin, xmax, ymax;
      int depth;
      int numcolors;
      struct gdrawfunctions *drawfunctions;
```

```
GT_FNPTR init, termin, open, close;
int videomode;
char *driverdata;
```

The following variables are usually used by and can be inquired by the actual video driver:

buffer

Address of the video buffer(if any). If NULL then a buffer is allocated internally for the size determined by the given device dimensions and depth parameters using the formulae:

size=(((xmax - xmin) * depth + 7) >> 3) * (ymax - ymin).

xmin, ymin, xmax, ymax

The device coordinates of the lower left hand corner and the upper right hand corner of the video device.

depth

The number of bits per pixel.

numcolors

The number of colors. This variable along with the given 'depth' and 'xmin, ymin, xmax, ymax' determine the byte width and overall size of the buffer.

drawfunctions

The address of an array of function pointers that are to handle the low level drawing for this video hardware device.

```
void init(videodriver)
GT_DRIVER videodriver;
```

The address of the function responsible for the initialization of the video hardware device.

```
void termin(videodriver)
GT_DRIVER videodriver;
```

The address of the function responsible for the termination of operations on the video hardware device. This function is called when graphics are terminated by a Gtermin() function call or when a new videodriver device is installed in a device graphic object.

```
void open(videodriver)
```

GT_DRIVER videodriver;

This is called whenever any graphic device object this videodriver is associated with is opened (i.e. previous draws or transformations were to a different graphic object device).

void close(videodriver)
GT_DRIVER videodriver;

This is called whenever any graphic device object this videodriver is associated with will no longer be used until the next 'open' function call.

videomode

An identification integer that can be used by the video driver. Ignored by PGL.

driverdata

A pointer the video driver may use if the videodriver data is insufficient.

EXAMPLES

BUGS

The close function is not used at this time.

SEE ALSO

Ginqvideodriver, Gmodvideodriver, Gdeletevideodriver.

Ginqvideodriver PGL LOW LEVEL INTERFACE Ginqvideodriver

DEFINITION

Inquires about the values of internal data in a videodriver device object.

PURPOSE

To provide a method by which video hardware device drivers can determine information about their driver state. (I.E. some drivers support multiple models of their device, such as the pixmap driver which supports all sizes of pixmaps in RAM).

OPERATION

void Ginqvideodriver(videodriver, GARG_keyword, &value,...)
GT_DRIVER videodriver;
int keyword;

*value;

The following variables can be obtained by using the following GARG keywords:

buffer GARG_BUFFER

Address of the video buffer(if any).

xmin, ymin, xmax, ymaxGARG_DEVICEDIMENSIONS

The device coordinates of the lower left hand corner and the upper right hand corner of the video device.

depth GARG_DEPTH

The number of bits per pixel.

numcolors GARG_MAXNUMCOLORS

The number of colors. This variable along with the given 'depth' and 'xmin, ymin, xmax, ymax' determine the byte width and overall size of the buffer.

videomode GARG_VIDEOMODE

An identification integer that can be used by the video driver. Ignored by PGL.

driverdata GARG_USERINFO

A pointer the video driver may use if the videodriver data is insufficient.

EXAMPLES

BUGS

The drawfunctions, init, termin, open, close parameters cannot be inquired at this time.

SEE ALSO

Gvideodriver, Gmodvideodriver, Gdeletevideodriver.

Gmodvideodriver PGL LOW LEVEL INTERFACE Gmodvideodriver

DEFINITION

Modifies the values of internal data in a videodriver device object.

PURPOSE

To provide a method by which video hardware device drivers can tailor information about their driver state.

OPERATION

```
void Gmodvideodriver(videodriver, GARG_keyword, value,...)
GT_DRIVER videodriver;
int keyword;
value;
```

The following variables can be modified by using the following GARG keywords:

buffer GARG_BUFFER

Address of the video buffer(if any).

xmin, ymin, xmax, ymaxGARG_DEVICEDIMENSIONS

The device coordinates of the lower left hand corner and the upper right hand corner of the video device.

depth GARG_DEPTH

The number of bits per pixel.

numcolors GARG_MAXNUMCOLORS

The number of colors. This variable along with the given 'depth' and 'xmin, ymin, xmax, ymax' determine the byte width and overall size of the buffer.

videomode GARG_VIDEOMODE

An identification integer that can be used by the video driver. Ignored by PGL.

driverdata GARG_USERINFO

A pointer the video driver may use if the videodriver data is insufficient.

EXAMPLES

BUGS

The drawfunctions, init, termin, open, close parameters cannot be modified at this time.
If the device dimensions change and the buffer was allocated internally the buffer is not freed and reallocated.

SEE ALSO

Gvideodriver, Ginqvideodriver, Gdeletevideodriver.

Gdeletevideodriver PGL LOW LEVEL INTERFACE Gdeletevideodriver

DEFINITION

Deletes any memory that has been allocated to support the given videodriver object. This includes the video 'buffer' if it was allocated internally during the given videodriver object's creation.

PURPOSE

To provide a method by which video hardware objects can be destroyed.

OPERATION

```
void Gdeletevideodriver(videodriver)
GT_DRIVER videodriver;
```

EXAMPLES

BUGS

SEE ALSO

Gvideodriver, Ginqvideodriver, Gmodvideodriver.

PGL LOW LEVEL INTERFACE

Application window drivers are also supported.

#include "graphics.h"

The following are examples of code that create the classical text string 'hello world' in a window.

```
/************************************************************/
main()
      {
      GT_OBJECT window;
      GT_OBJECT view;
      GT_OBJECT picture;

      Ginit();
      Gdevice(0);
```

```
      window = Gwindow(GARG_SCREEN, G_MINSCREEN, G_MINSCREEN, G_MAXSCREEN,
G_MAXSCREEN, 0);
      view = Gview(GARG_VIEWPORT, G_MINVIEWPORT, G_MINVIEWPORT, G_MAXVIEWPORT,
G_MAXVIEWPORT, GARG_WORLD, G_MINWORLD, G_MINWORLD, G_MAXWORLD, G_MAXWORLD, 0);
      picture = Gpicture(view, window, G_OPENOBJECT);
      Gopen(picture);
      Gdrawtext((GT_COORD )0, (GT_COORD )0, "Hello World");
      Gtermin();
      }

/*************************************************************/
main()
      {
      GT_OBJECT window;
      GT_OBJECT view;
      GT_OBJECT picture;
      GT_OBJECT segment;

      Ginit();
      Gdevice(0);
      window = Gwindow(GARG_SCREEN, G_MINSCREEN, G_MINSCREEN, G_MAXSCREEN,
G_MAXSCREEN, 0);
      segment = Gsegment(0);
      Gopen(segment);
      Gtext((GT_COORD )0, (GT_COORD )0, "Hello World");
      view = Gview(
            GARG_VIEWPORT,
               G_MINVIEWPORT, G_MINVIEWPORT, G_MAXVIEWPORT, G_MAXVIEWPORT,
            GARG_WORLD,
               G_MINWORLD, G_MINWORLD, G_MAXWORLD, G_MAXWORLD,
            GARG_SEGMENT,
               segment,
            0);
      picture = Gpicture(view, window, G_OPENOBJECT);
      Gdraw(picture);
      Gtermin();
      }

/*************************************************************/
main()
      {
      GT_OBJECT window;
      GT_OBJECT view;
      GT_OBJECT picture;
      GT_OBJECT segment;

      Ginit();
      Gdevice(0);
      window = Gwindow(
            GARG_SCREEN,
                  G_MINSCREEN, G_MINSCREEN, G_MAXSCREEN, G_MAXSCREEN,
            0);
      segment = Gsegment(0);
      Gopen(segment);
```

```
      Gtext((GT_COORD )0, (GT_COORD )0, "Hello World");
      view = Gview(
            GARG_VIEWPORT,
               G_MINVIEWPORT, G_MINVIEWPORT, G_MAXVIEWPORT, G_MAXVIEWPORT,
            GARG_WORLD,
               G_MINWORLD, G_MINWORLD, G_MAXWORLD, G_MAXWORLD,
            GARG_SEGMENT,
               segment,
            GARG_PROC,
                  our_window_event_handler,
            0);
      picture = Gpicture(view, window, G_OPENOBJECT);
      Gdraw(picture);
      Gwaitforevent(G_KEYEVENT);
      Gtermin();
      }

#define our_zoominmaskG_M_CLICK
#define our_zoominkeyG_M_CLICK
#define our_zoomoutmaskG_R_CLICK
#define our_zoomoutkeyG_R_CLICK
#define our_panmaskG_M_WENTDOWN
#define our_pankeyG_M_WENTDOWN

static int our_window_event_handler(event)
GT_EVENT event;
      {
      if ((Gevent_type(event) & our_zoominmask)
            && (Gevent_value(event) == our_zoominkey))
            {
            Gzoom(view, x, y, GF_ZOOMSHIFT, G_ZOOMIN);
            return(0);
            }
      if ((Gevent_type(event) & our_zoomoutmask)
            && (Gevent_value(event) == our_zoomoutkey))
            {
            Gzoom(view, x, y, GF_ZOOMSHIFT, G_ZOOMOUT);
            return(0);
            }
      if ((Gevent_type(event) & our_panmask)
            && (Gevent_value(event) == our_pankey))
            {
            Gpan(Gevent_view(event), G_MINWORLD,
                  G_MINWORLD,
                  G_MAXWORLD,
                  G_MAXWORLD);
            return(0);
            }
      return(1);
      }

#include "toolkit.h"
```

```
/*****************************************************************/
main()
      {
      GFT_OBJECT form;

      Ginit();
      Gdevice(0);
      GFinit();
      form = GFform(
            TKARG_SCREEN,
                   G_MINSCREEN, G_MINSCREEN, G_MAXSCREEN, G_MAXSCREEN,
            TKARG_ZOOMINEVENT,
                  our_zoominmask, our_zoominkey,
            TKARG_ZOOMOUTEVENT,
                  our_zoomoutmask, our_zoomoutkey,
            TKARG_PANEVENT,
                  our_panmask, our_pankey,
            0);
      GFbutton(form,
             TKARG_DESCX, 0,
             TKARG_DESCY, 0,
             TKARG_DESCTEXT, "HelloWorld",
             TKARG_PROC, our_button_event_handler,
             0);
      GFpost(form);
      Gwaitforevent(G_L_CLICK);
      GFtermin();
      Gtermin();
      }
static int our_button_event_handler(button, event)
GFT_OBJECT button;
GT_EVENT event;
      {
      return(1);
      }
/*****************************************************************/
```

PGL QUESTIONAIRE

Please answer this questionaire if you have an opinion and the time.

1. make array structs have type so don't need compile funcs?

2. make wascii check ctxdiffs so it can be a function?

3. make setobj and transobj update extrema so they can be called too?

4. Should the cursor routines take a picture object as a parameter rather than using the current open picture?

Version 1.0 Additions

1. The functions Gmodarc, Gmodcall, Gmodellipse, Gmodline, Gmodpline, Gmodpolygon, Gmodrect, and Gmodpolygon have been added. Also the Guserobj object and support functions Gmoddraw, Ginqdraw, and Gmodtransform, Ginqtransform.

2. Library size has been reduced by removing the 'C' source code write/read functionality and by removing the object description (struct gdebug) inquire functionality. These are available in the new PGL toolkit library.

3. Bugs have been fixed.