Nano-X API Reference Manual

Nano-X API Reference Manual

Table of Contents

1. libnano-X 4
general 4
window 7
graphics 17
events 41
fonts 44
pointer 47
colours 50
regions 53
selections 61
misc 65

Chapter 1. libnano-X

general (3)

Name

`general` —

Synopsis

```
void        GrFlush                         (void);
int         GrOpen                          (void);
void        GrClose                         (void);
void        GrMain-
Loop                 (GR_FNCALLBACKEVENT fncb);
void        GrGetScreen-
Info            (GR_SCREEN_INFO *sip);
GR_FNCALLBACKEVENT GrSetErrorHan-
dler        (GR_FNCALLBACKEVENT fncb);
void        GrDefaultErrorHandler           (GR_EVENT *ep);
```

Description

Details

GrFlush ()

```
void        GrFlush                        (void);
```

Flush the message buffer of any messages it may contain.

GrOpen ()

```
int         GrOpen                         (void);
```

Open a connection to the graphics server.

Returns : the fd of the connection to the server or -1 on failure

GrClose ()

```
void        GrClose                        (void);
```

Close the graphics device, flushing any waiting messages.

GrMainLoop ()

```
void        GrMain-
Loop                         (GR_FNCALLBACKEVENT fncb);
```

A convenience function which calls the specified callback function whenever an event arrives or there is data to be read on a file descriptor previously specified by GrRegisterInput(). Currently never returns.

`fncb` :

GrGetScreenInfo ()

```
void        GrGetScreen-
Info                 (GR_SCREEN_INFO *sip);
```

Fills in the specified GR_SCREEN_INFO structure.

`sip` : pointer to a GR_SCREEN_INFO structure

GrSetErrorHandler ()

```
GR_FNCALLBACKEVENT GrSetErrorHan-
dler        (GR_FNCALLBACKEVENT fncb);
```

Sets an error handling routine that will be called on any errors from the server (assuming the client has asked to receive them). If zero is used as the argument, errors will be returned as regular events instead.

`fncb` : the function to call to handle error events

Returns : the address of the previous error handler

GrDefaultErrorHandler ()

```
void        GrDefaultErrorHandler           (GR_EVENT *ep);
```

The default error handler which is called when the server reports an error event and the client hasn't set up a handler of it's own.

Generates a human readable error message on stderr describing what error occurred and what function it occured in, then exits.

`ep` : the error event structure

window (3)

Name

`window` —

Synopsis

```
GR_WINDOW_ID GrNewWin-
dow                    (GR_WINDOW_ID parent,
                                              GR_COORD x,
                                              GR_COORD y,
                                              GR_SIZE width,
                                              GR_SIZE height,
                                              GR_SIZE bordersize,
                                              GR_COLOR background,
                                              GR_COLOR bordercolor);
GR_WINDOW_ID GrNewPixmap                     (GR_SIZE width,
                                              GR_SIZE height,
                                              void *addr);
GR_WINDOW_ID GrNewInputWin-
dow               (GR_WINDOW_ID parent,
                                                      GR_COORD x,
                                                      GR_COORD y,
                                                      GR_SIZE width,
                                                      GR_SIZE height);
void         GrDestroyWindow                         (GR_WINDOW_ID wid);
void         GrMapWindow                             (GR_WINDOW_ID wid);
void         GrUnmapWindow                           (GR_WINDOW_ID wid);
void         GrRaiseWindow                           (GR_WINDOW_ID wid);
void         GrLowerWindow                           (GR_WINDOW_ID wid);
void         GrMoveWindow                            (GR_WINDOW_ID wid,
                                                      GR_COORD x,
                                                      GR_COORD y);
void         GrResizeWindow                          (GR_WINDOW_ID wid,
                                                      GR_SIZE width,
                                                      GR_SIZE height);
void         GrReparentWindow                        (GR_WINDOW_ID wid,
                                                      GR_WINDOW_ID pwid,
                                                      GR_COORD x,
                                                      GR_COORD y);
void         GrGetWindowInfo                         (GR_WINDOW_ID wid,
                                                      GR_WINDOW_INFO *infoptr);
void         GrSetWMProperties                       (GR_WINDOW_ID wid,
                                                      GR_WM_PROPERTIES *props);
void         GrGetWMProperties                       (GR_WINDOW_ID wid,
                                                      GR_WM_PROPERTIES *props);
void         GrSetFocus                              (GR_WINDOW_ID wid);
GR_WINDOW_ID GrGetFocus                              (void);
void         GrSetBorderColor                        (GR_WINDOW_ID wid,
                                                      GR_COLOR color);
void         GrSetBackgroundPixmap                   (GR_WINDOW_ID wid,
                                                      GR_WINDOW_ID pixmap,
                                                      int flags);
void         GrClearWindow                           (GR_WINDOW_ID wid,
                                                      GR_BOOL exposeflag);
void         GrCloseWindow                           (GR_WINDOW_ID wid);
void         GrKillWindow                            (GR_WINDOW_ID wid);
```

Description

Details

GrNewWindow ()

```
GR_WINDOW_ID GrNewWindow                   (GR_WINDOW_ID parent,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE width,
                                             GR_SIZE height,
                                             GR_SIZE bordersize,
                                             GR_COLOR background,
                                             GR_COLOR bordercolor);
```

Create a new window with the specified parent and window attributes.

`parent` :	the ID of the parent window
`x` :	the X coordinate of the new window relative to the parent window
`y` :	the Y coordinate of the new window relative to the parent window
`width` :	the width of the new window
`height` :	the height of the new window
`bordersize` :	the width of the window border
`background` :	the colour of the window background
`bordercolor` :	the colour of the window border

Returns :	the ID of the newly created window

GrNewPixmap ()

```
GR_WINDOW_ID GrNewPixmap                    (GR_SIZE width,
                                             GR_SIZE height,
                                             void *addr);
```

Create a new server side pixmap (an offscreen drawing area which can be copied into a window using a GrCopyArea call) of the specified width and height.

`width` :	the width of the pixmap
`height` :	the height of the pixmap
`addr` :	currently unused in client/server mode
Returns :	the ID of the newly created pixmap

GrNewInputWindow ()

```
GR_WINDOW_ID GrNewInputWindow               (GR_WINDOW_ID par-
ent,

                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE width,
                                             GR_SIZE height);
```

Create a new input-only window with the specified dimensions which is a child of the specified parent window.

`parent` :	the ID of the window to use as the parent of the new window
`x` :	the X coordinate of the new window relative to the parent window

`y` : the Y coordinate of the new window relative to the parent window
`width` : the width of the new window
`height` : the height of the new window
Returns : the ID of the newly created window

GrDestroyWindow ()

```
void        GrDestroyWindow                  (GR_WINDOW_ID wid);
```

Recursively unmaps and frees the data structures associated with the specified window and all of its children.

`wid` : the ID of the window to destroy

GrMapWindow ()

```
void        GrMapWindow                      (GR_WINDOW_ID wid);
```

Recursively maps (makes visible) the specified window and all of the child windows which have a sufficient map count. The border and background of the window are painted, and an exposure event is generated for the window and every child which becomes visible.

`wid` : the ID of the window to map

GrUnmapWindow ()

```
void        GrUnmapWindow                    (GR_WINDOW_ID wid);
```

Recursively unmaps (makes invisible) the specified window and all of the child windows.

`wid` : the ID of the window to unmap

GrRaiseWindow ()

```
void        GrRaiseWindow                  (GR_WINDOW_ID wid);
```

Places the specified window at the top of its parents drawing stack, above all of its sibling windows.

`wid` : the ID of the window to raise

GrLowerWindow ()

```
void        GrLowerWindow                  (GR_WINDOW_ID wid);
```

Places the specified window at the bottom of its parents drawing stack, below all of its sibling windows.

`wid` : the ID of the window to lower

GrMoveWindow ()

```
void        GrMoveWindow                   (GR_WINDOW_ID wid,
                                            GR_COORD x,
                                            GR_COORD y);
```

Moves the specified window to the specified position relative to its parent window.

`wid` : the ID of the window to move

`x` : the X coordinate to move the window to relative to its parent.

`y` : the Y coordinate to move the window to relative to its parent.

GrResizeWindow ()

```
void        GrResizeWindow                  (GR_WINDOW_ID wid,
                                             GR_SIZE width,
                                             GR_SIZE height);
```

Resizes the specified window to be the specified width and height.

`wid` : the ID of the window to resize

`width` : the width to resize the window to

`height` : the height to resize the window to

GrReparentWindow ()

```
void        GrReparentWindow                (GR_WINDOW_ID wid,
                                             GR_WINDOW_ID pwid,
                                             GR_COORD x,
                                             GR_COORD y);
```

Changes the parent window of the specified window to the specified parent window and places it at the specified coordinates relative to the new parent.

`wid` : the ID of the window to reparent

`pwid` : the ID of the new parent window
`x` : the X coordinate to place the window at relative to the new parent
`y` : the Y coordinate to place the window at relative to the new parent

GrGetWindowInfo ()

```
void        GrGetWindowInfo                 (GR_WINDOW_ID wid,
                                             GR_WINDOW_INFO *in-
foptr);
```

Fills in a GR_WINDOW_INFO structure with information regarding the window with the specified window ID.

`wid` : the ID of the window to retrieve information about
`infoptr` : pointer to a GR_WINDOW_INFO structure to return the information in

GrSetWMProperties ()

```
void        GrSetWMProperties               (GR_WINDOW_ID wid,
                                             GR_WM_PROPERTIES *props);
```

Copies the provided GR_WM_PROPERTIES structure into the the GR_WM_PROPERTIES structure of the specified window id.

`wid` : the ID of the window to set the WM properties of
`props` : pointer to a GR_WM_PROPERTIES structure

GrGetWMProperties ()

```
void        GrGetWMProperties               (GR_WINDOW_ID wid,
                                             GR_WM_PROPERTIES *props);
```

Reads the GR_WM_PROPERTIES structure for the window with the specified id and fills in the provided structure with the information. It is the callers responsibility to free the title member as it is allocated dynamically. The title field will be set to NULL if the window has no title.

`wid` :	the ID of the window to retreive the WM properties of
`props` :	pointer to a GR_WM_PROPERTIES structure to fill in

GrSetFocus ()

```
void        GrSetFocus                      (GR_WINDOW_ID wid);
```

Sets the keyboard focus to the specified window.

`wid` :	the ID of the window to set the focus to

GrGetFocus ()

```
GR_WINDOW_ID GrGetFocus                     (void);
```

Returns :	the ID of the window which currently has the keyboard focus

GrSetBorderColor ()

```
void        GrSetBorderColor                (GR_WINDOW_ID wid,
                                             GR_COLOR color);
```

Sets the border colour of the specified window to the specified colour.

wid : the ID of the window to set the border colour of

color :

GrSetBackgroundPixmap ()

```
void        GrSetBackgroundPixmap           (GR_WINDOW_ID wid,
                                             GR_WINDOW_ID pixmap,
                                             int flags);
```

Sets the background of the specified window to the specified pixmap. The flags which specify how to draw the pixmap (in the top left of the window, in the centre of the window, tiled, etc.) are those which start with GR_BACKGROUND_ in nano-X.h. If the pixmap value is 0, the server will disable the background pixmap and return to using a solid colour fill.

wid : ID of the window to set the background of

pixmap : ID of the pixmap to use as the background

flags : flags specifying how to draw the pixmap onto the window

GrClearWindow ()

```
void        GrClearWindow                   (GR_WINDOW_ID wid,
                                             GR_BOOL expose-
flag);
```

Clears the specified window by setting it to its background color. If the exposeflag parameter is non zero, an exposure event is generated for the window after it has been cleared.

`wid` : the ID of the window to clear

`exposeflag` : a flag indicating whether to also generate an exposure event

GrCloseWindow ()

```
void        GrCloseWindow                   (GR_WINDOW_ID wid);
```

Sends a CLOSE_REQ event to the specified window if the client has selected to receive CLOSE_REQ events on this window. Used to request an application to shut down but not force it to do so immediately, so the application can ask whether to save changed files before shutting down cleanly.

`wid` : the ID of the window to send the CLOSE_REQ event to

GrKillWindow ()

```
void        GrKillWindow                    (GR_WINDOW_ID wid);
```

Forcibly disconnects the client which owns this window with the specified ID number. Used to kill an application which has locked up and is not responding to CLOSE_REQ events.

`wid` : the ID of the window to kill

graphics (3)

Name

`graphics` —

Synopsis

```
GR_GC_ID    GrNewGC                                 (void);
GR_GC_ID    GrCopyGC                                (GR_GC_ID gc);
void        GrGetGCInfo                             (GR_GC_ID gc,
                                                     GR_GC_INFO *gcip);
void        GrDestroyGC                             (GR_GC_ID gc);
void        GrLine                                  (GR_DRAW_ID id,
                                                     GR_GC_ID gc,
                                                     GR_COORD x1,
                                                     GR_COORD y1,
                                                     GR_COORD x2,
                                                     GR_COORD y2);
void        GrPoint                                 (GR_DRAW_ID id,
                                                     GR_GC_ID gc,
                                                     GR_COORD x,
                                                     GR_COORD y);
void        GrPoints                                (GR_DRAW_ID id,
                                                     GR_GC_ID gc,
                                                     GR_COUNT count,
                                                     GR_POINT *pointtable);
void        GrRect                                  (GR_DRAW_ID id,
                                                     GR_GC_ID gc,
                                                     GR_COORD x,
                                                     GR_COORD y,
                                                     GR_SIZE width,
                                                  GR_SIZE height);
void         GrFillRect                          (GR_DRAW_ID id,
                                                  GR_GC_ID gc,
                                                  GR_COORD x,
                                                  GR_COORD y,
                                                  GR_SIZE width,
                                                  GR_SIZE height);
void         GrPoly                              (GR_DRAW_ID id,
                                                  GR_GC_ID gc,
                                                  GR_COUNT count,
                                                  GR_POINT *pointtable);
void         GrFillPoly                          (GR_DRAW_ID id,
                                                  GR_GC_ID gc,
                                                  GR_COUNT count,
                                                  GR_POINT *pointtable);
void         GrEllipse                           (GR_DRAW_ID id,
                                                  GR_GC_ID gc,
                                                  GR_COORD x,
                                                  GR_COORD y,
                                                  GR_SIZE rx,
                                                  GR_SIZE ry);
void         GrFillEllipse                       (GR_DRAW_ID id,
                                                  GR_GC_ID gc,
                                                  GR_COORD x,
                                                  GR_COORD y,
                                                  GR_SIZE rx,
                                                  GR_SIZE ry);
void         GrArc                               (GR_DRAW_ID id,
                                                  GR_GC_ID gc,
                                                  GR_COORD x,
                                                  GR_COORD y,
                                                  GR_SIZE rx,
                                                  GR_SIZE ry,
                                                  GR_COORD ax,
                                                  GR_COORD ay,
                                                  GR_COORD bx,
                                                  GR_COORD by,
                                             int type);
void        GrArcAngle                      (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE rx,
                                             GR_SIZE ry,
                                             GR_COORD angle1,
                                             GR_COORD angle2,
                                             int type);
void        GrSetGCForeground               (GR_GC_ID gc,
                                             GR_COLOR foreground);
void        GrSetGCBackground               (GR_GC_ID gc,
                                             GR_COLOR background);
void        GrSetGCUseBackground            (GR_GC_ID gc,
                                             GR_BOOL flag);
void        GrSetGCMode                     (GR_GC_ID gc,
                                             int mode);
void        GrSetGCFont                     (GR_GC_ID gc,
                                             GR_FONT_ID font);
void        GrGetGCTextSize                 (GR_GC_ID gc,
                                             void *str,
                                             int count,
                                             int flags,
                                             GR_SIZE *retwidth,
                                             GR_SIZE *retheight,
                                             GR_SIZE *retbase);
void        GrReadArea                      (GR_DRAW_ID id,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE width,
                                             GR_SIZE height,
                                             GR_PIXELVAL *pixels);
void        GrArea                          (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                           GR_SIZE width,
                                           GR_SIZE height,
                                           void *pixels,
                                           int pixtype);
void        GrCopyArea                    (GR_DRAW_ID id,
                                           GR_GC_ID gc,
                                           GR_COORD x,
                                           GR_COORD y,
                                           GR_SIZE width,
                                           GR_SIZE height,
                                           GR_DRAW_ID srcid,
                                           GR_COORD srcx,
                                           GR_COORD srcy,
                                           int op);
void        GrBitmap                      (GR_DRAW_ID id,
                                           GR_GC_ID gc,
                                           GR_COORD x,
                                           GR_COORD y,
                                           GR_SIZE width,
                                           GR_SIZE height,
                                           GR_BITMAP *imagebits);
void        GrFreeImage                   (GR_IMAGE_ID id);
void        GrGetImageInfo                (GR_IMAGE_ID id,
                                           GR_IMAGE_INFO *iip);
void        GrDrawImageFromFile           (GR_DRAW_ID id,
                                           GR_GC_ID gc,
                                           GR_COORD x,
                                           GR_COORD y,
                                           GR_SIZE width,
                                           GR_SIZE height,
                                           char *path,
                                           int flags);
GR_IMAGE_ID GrLoadImageFromFile           (char *path,
                                           int flags);
void        GrDrawImageToFit              (GR_DRAW_ID id,
                                           GR_GC_ID gc,
                                           GR_COORD x,
                                            GR_COORD y,
                                            GR_SIZE width,
                                            GR_SIZE height,
                                            GR_IMAGE_ID imageid);
void        GrDrawImageBits                (GR_DRAW_ID id,
                                            GR_GC_ID gc,
                                            GR_COORD x,
                                            GR_COORD y,
                                            GR_IMAGE_HDR *pimage);
void        GrText                         (GR_DRAW_ID id,
                                            GR_GC_ID gc,
                                            GR_COORD x,
                                            GR_COORD y,
                                            void *str,
                                            GR_COUNT count,
                                            int flags);
```

Description

Details

GrNewGC ()

```
GR_GC_ID    GrNewGC                        (void);
```

Creates a new graphics context structure and returns the ID used to refer to it. The structure is initialised with a set of default parameters.

Returns : the ID of the newly created graphics context or 0 on error

GrCopyGC ()

```
GR_GC_ID    GrCopyGC                        (GR_GC_ID gc);
```

Creates a new graphics context structure and fills it in with the values from the specified already existing graphics context.

`gc` :	the already existing graphics context to copy the parameters from
Returns :	the ID of the newly created graphics context or 0 on error

GrGetGCInfo ()

```
void        GrGetGCInfo                     (GR_GC_ID gc,
                                             GR_GC_INFO *gcip);
```

Fills in the specified GR_GC_INFO structure with information regarding the specified graphics context.

`gc` :	a graphics context
`gcip` :	pointer to a GR_GC_INFO structure

GrDestroyGC ()

```
void        GrDestroyGC                     (GR_GC_ID gc);
```

Destroys the graphics context structure with the specified ID.

`gc` :	the ID of the graphics context structure to destroy

GrLine ()

```
void        GrLine                          (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x1,
                                             GR_COORD y1,
                                             GR_COORD x2,
                                             GR_COORD y2);
```

Draws a line using the specified graphics context on the specified drawable from (x1, y1) to (x2, y2), with coordinates given relative to the drawable.

id :	the ID of the drawable to draw the line on
gc :	the ID of the graphics context to use when drawing the line
x1 :	the X coordinate of the start of the line relative to the drawable
y1 :	the Y coordinate of the start of the line relative to the drawable
x2 :	the X coordinate of the end of the line relative to the drawable
y2 :	the Y coordinate of the end of the line relative to the drawable

GrPoint ()

```
void        GrPoint                         (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y);
```

Draws a point using the specified graphics context at the specified position on the specified drawable.

id :	the ID of the drawable to draw a point on
gc :	the ID of the graphics context to use when drawing the point

`x` : the X coordinate to draw the point at relative to the drawable

`y` : the Y coordinate to draw the point at relative to the drawable

GrPoints ()

```
void        GrPoints                        (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COUNT count,
                                             GR_POINT *point-
table);
```

Draws a set of points using the specified graphics context at the positions specified by the point table on the specified drawable.

`id` : the ID of the drawable to draw a point on

`gc` : the ID of the graphics context to use when drawing the point

`count` : the number of points in the point table

`pointtable` : pointer to a GR_POINT array which lists the points to draw

GrRect ()

```
void        GrRect                          (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE width,
                                             GR_SIZE height);
```

Draw the boundary of a rectangle of the specified dimensions and position on the specified drawable using the specified graphics context.

`id` : the ID of the drawable to draw the rectangle on
`gc` : the ID of the graphics context to use when drawing the rectangle
`x` : the X coordinate of the rectangle relative to the drawable
`y` : the Y coordinate of the rectangle relative to the drawable
`width` : the width of the rectangle
`height` : the height of the rectangle

GrFillRect ()

```
void        GrFillRect                      (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE width,
                                             GR_SIZE height);
```

Draw a filled rectangle of the specified dimensions and position on the specified drawable using the specified graphics context.

`id` : the ID of the drawable to draw the rectangle on
`gc` : the ID of the graphics context to use when drawing the rectangle
`x` : the X coordinate of the rectangle relative to the drawable
`y` : the Y coordinate of the rectangle relative to the drawable
`width` : the width of the rectangle
`height` : the height of the rectangle

GrPoly ()

```
void        GrPoly                          (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COUNT count,
                                             GR_POINT *point-
table);
```

Draws an unfilled polygon on the specified drawable using the specified graphics context. The polygon is specified by an array of point structures. The polygon is not automatically closed- if a closed polygon is desired, the last point must be the same as the first.

`id` : the ID of the drawable to draw the polygon onto
`gc` : the ID of the graphics context to use when drawing the polygon
`count` : the number of points in the point array
`pointtable` : pointer to an array of points describing the polygon

GrFillPoly ()

```
void        GrFillPoly                      (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COUNT count,
                                             GR_POINT *point-
table);
```

Draws a filled polygon on the specified drawable using the specified graphics context. The polygon is specified by an array of point structures. The polygon is automatically closed- the last point need not be the same as the first in order for the polygon to be closed.

`id` : the ID of the drawable to draw the polygon onto
`gc` : the ID of the graphics context to use when drawing the polygon
`count` : the number of points in the point array

`pointtable` :	pointer to an array of points describing the polygon

GrEllipse ()

```
void        GrEllipse                        (GR_DRAW_ID id,
                                              GR_GC_ID gc,
                                              GR_COORD x,
                                              GR_COORD y,
                                              GR_SIZE rx,
                                              GR_SIZE ry);
```

Draws the boundary of ellipse at the specified position using the specified dimensions and graphics context on the specified drawable.

`id` :	the ID of the drawable to draw the ellipse on
`gc` :	the ID of the graphics context to use when drawing the ellipse
`x` :	the X coordinate to draw the ellipse at relative to the drawable
`y` :	the Y coordinate to draw the ellipse at relative to the drawable
`rx` :	the radius of the ellipse on the X axis
`ry` :	the radius of the ellipse on the Y axis

GrFillEllipse ()

```
void        GrFillEllipse                    (GR_DRAW_ID id,
                                              GR_GC_ID gc,
                                              GR_COORD x,
                                              GR_COORD y,
                                              GR_SIZE rx,
                                              GR_SIZE ry);
```

Draws a filled ellipse at the specified position using the specified dimensions and graphics context on the specified drawable.

`id` : the ID of the drawable to draw the filled ellipse on
`gc` : the ID of the graphics context to use when drawing the ellipse
`x` : the X coordinate to draw the ellipse at relative to the drawable
`y` : the Y coordinate to draw the ellipse at relative to the drawable
`rx` : the radius of the ellipse on the X axis
`ry` : the radius of the ellipse on the Y axis

GrArc ()

```
void        GrArc                           (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE rx,
                                             GR_SIZE ry,
                                             GR_COORD ax,
                                             GR_COORD ay,
                                             GR_COORD bx,
                                             GR_COORD by,
                                             int type);
```

Draws an arc with the specified dimensions at the specified position on the specified drawable using the specified graphics context. The type specifies the fill type. Possible values include GR_ARC and GR_PIE.

`id` : the ID of the drawable to draw the arc on
`gc` : the graphics context to use when drawing the arc
`x` : the X coordinate to draw the arc at relative to the drawable

`y` : the Y coordinate to draw the arc at relative to the drawable
`rx` : the radius of the arc on the X axis
`ry` : the radius of the arc on the Y axis
`ax` : the X coordinate of the start of the arc relative to the drawable
`ay` : the Y coordinate of the start of the arc relative to the drawable
`bx` : the X coordinate of the end of the arc relative to the drawable
`by` : the Y coordinate of the end of the arc relative to the drawable
`type` : the fill style to use when drawing the arc

GrArcAngle ()

```
void        GrArcAngle                      (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE rx,
                                             GR_SIZE ry,
                                             GR_COORD angle1,
                                             GR_COORD angle2,
                                             int type);
```

Draws an arc with the specified dimensions at the specified position on the specified drawable using the specified graphics context. The type specifies the fill type. Possible values include GR_ARC and GR_PIE. This function requires floating point support, and is slightly slower than the GrArc() function which does not require floating point.

`id` : the ID of the drawable to draw the arc on
`gc` : the graphics context to use when drawing the arc
`x` : the X coordinate to draw the arc at relative to the drawable
`y` : the Y coordinate to draw the arc at relative to the drawable

`rx` : the radius of the arc on the X axis
`ry` : the radius of the arc on the Y axis
`angle1` : the angle of the start of the arc
`angle2` : the angle of the end of the arc
`type` : the fill style to use when drawing the arc

GrSetGCForeground ()

```
void        GrSetGCForeground              (GR_GC_ID gc,
                                             GR_COLOR fore-
ground);
```

Changes the foreground colour of the specified graphics context to the specified colour.

`gc` : the ID of the graphics context to set the foreground colour of
`foreground` : the colour to use as the new foreground colour

GrSetGCBackground ()

```
void        GrSetGCBackground              (GR_GC_ID gc,
                                             GR_COLOR back-
ground);
```

Changes the background colour of the specified graphics context to the specified colour.

`gc` : the ID of the graphics context to set the background colour of
`background` : the colour to use as the new background colour

GrSetGCUseBackground ()

```
void        GrSetGCUseBackground            (GR_GC_ID gc,
                                             GR_BOOL flag);
```

Sets the flag which chooses whether or not the background colour is used when drawing bitmaps and text using the specified graphics context to the specified value.

gc :	the ID of the graphics context to change the "use background" flag of
flag :	flag specifying whether to use the background colour or not

GrSetGCMode ()

```
void        GrSetGCMode                     (GR_GC_ID gc,
                                             int mode);
```

Changes the drawing mode (SET, XOR, OR, AND, etc.) of the specified graphics context to the specified mode.

gc :	the ID of the graphics context to set the drawing mode of
mode :	the new drawing mode

GrSetGCFont ()

```
void        GrSetGCFont                     (GR_GC_ID gc,
                                             GR_FONT_ID font);
```

Sets the font to be used for text drawing in the specified graphics context to the specified font ID.

`gc` : the ID of the graphics context to set the font of
`font` : the ID of the font

GrGetGCTextSize ()

```
void        GrGetGCTextSize                 (GR_GC_ID gc,
                                             void *str,
                                             int count,
                                             int flags,
                                             GR_SIZE *retwidth,
                                             GR_SIZE *retheight,
                                             GR_SIZE *retbase);
```

Calculates the dimensions of the specified text string using the current font and flags in the specified graphics context. The count argument can be -1 if the string is null terminated.

`gc` : the graphics context
`str` : pointer to a text string
`count` : the length of the string
`flags` : text rendering flags (GR_TF*)
`retwidth` : pointer to the variable the width will be returned in
`retheight` : pointer to the variable the height will be returned in
`retbase` : pointer to the variable the baseline height will be returned in

GrReadArea ()

```
void        GrReadArea                      (GR_DRAW_ID id,
                                             GR_COORD x,
                                             GR_COORD y,
                                                  GR_SIZE width,
                                                  GR_SIZE height,
                                                  GR_PIXELVAL *pix-
els);
```

Reads the pixel data of the specified size from the specified position on the specified drawable into the specified pixel array. If the drawable is a window, the data returned will be the pixel values from the relevant position on the screen regardless of whether the window is obscured by other windows. If the window is unmapped, or partially or fully outside a window boundary, black pixel values will be returned.

`id` :	the ID of the drawable to read an area from
`x` :	the X coordinate to read the area from relative to the drawable
`y` :	the Y coordinate to read the area from relative to the drawable
`width` :	the width of the area to read
`height` :	the height of the area to read
`pixels` :	pointer to an area of memory to place the pixel data in

GrArea ()

```
void        GrArea                           (GR_DRAW_ID id,
                                              GR_GC_ID gc,
                                              GR_COORD x,
                                              GR_COORD y,
                                              GR_SIZE width,
                                              GR_SIZE height,
                                              void *pixels,
                                              int pixtype);
```

Draws the specified pixel array of the specified size and format onto the specified drawable using the specified graphics context at the specified position. Note that colour conversion is currently only performed when using the GR_PF_RGB format, which is an unsigned long containing RGBX data.

`id` :	the ID of the drawable to draw the area onto
`gc` :	the ID of the graphics context to use when drawing the area
`x` :	the X coordinate to draw the area at relative to the drawable
`y` :	the Y coordinate to draw the area at relative to the drawable
`width` :	the width of the area
`height` :	the height of the area
`pixels` :	pointer to an array containing the pixel data
`pixtype` :	the format of the pixel data

GrCopyArea ()

```
void        GrCopyArea                    (GR_DRAW_ID id,
                                            GR_GC_ID gc,
                                            GR_COORD x,
                                            GR_COORD y,
                                            GR_SIZE width,
                                            GR_SIZE height,
                                            GR_DRAW_ID srcid,
                                            GR_COORD srcx,
                                            GR_COORD srcy,
                                            int op);
```

Copies the specified area of the specified size between the specified drawables at the specified positions using the specified graphics context and ROP codes. 0 is a sensible default ROP code in most cases.

`id` :	the ID of the drawable to copy the area to
`gc` :	the ID of the graphics context to use when copying the area
`x` :	the X coordinate to copy the area to within the destination drawable

`y` : the Y coordinate to copy the area to within the destination drawable
`width` : the width of the area to copy
`height` : the height of the area to copy
`srcid` : the ID of the drawable to copy the area from
`srcx` : the X coordinate to copy the area from within the source drawable
`srcy` : the Y coordinate to copy the area from within the source drawable
`op` : the ROP codes to pass to the blitter when performing the copy

GrBitmap ()

```
void        GrBitmap                        (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE width,
                                             GR_SIZE height,
                                             GR_BITMAP *imagebits);
```

Draws the monochrome bitmap data provided in the bitmaptable argument at the specified position on the specified drawable using the specified graphics context. Note that the bitmap data should be an array of aligned 16 bit words. The usebackground flag in the graphics context specifies whether to draw the background colour wherever a bit value is zero.

`id` : the ID of the drawable to draw the bitmap onto
`gc` : the ID of the graphics context to use when drawing the bitmap
`x` : the X coordinate to draw the bitmap at relative to the drawable
`y` : the Y coordinate to draw the bitmap at relative to the drawable
`width` : the width of the bitmap

`height` : the height of the bitmap
`imagebits` :

GrFreeImage ()

```
void        GrFreeImage                     (GR_IMAGE_ID id);
```

Destroys the specified image buffer and reclaims the memory used by it.

`id` : ID of the image buffer to free

GrGetImageInfo ()

```
void        GrGetImageInfo                  (GR_IMAGE_ID id,
                                             GR_IMAGE_INFO *iip);
```

Fills in the specified image information structure with the details of the specified image buffer.

`id` : ID of an image buffer
`iip` : pointer to a GR_IMAGE_INFO structure

GrDrawImageFromFile ()

```
void        GrDrawImageFromFile             (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE width,
                                                       GR_SIZE height,
                                                       char *path,
                                                       int flags);
```

Loads the specified image file and draws it at the specified position on the specified drawable using the specified graphics context. The width and height values specify the size of the image to draw- if the actual image is a different size, it will be scaled to fit. The image type is automatically detected using the magic numbers in the image header (ie. the filename extension is irrelevant). The currently supported image types include GIF, JPEG, Windows BMP, PNG, XPM, and both ascii and binary variants of PBM, PGM, and PPM. However the image types supported by a particular server depend on which image types were enabled in the server configuration at build time.

`id` : the ID of the drawable to draw the image onto

`gc` : the ID of the graphics context to use when drawing the image

`x` : the X coordinate to draw the image at relative to the drawable

`y` : the Y coordinate to draw the image at relative to the drawable

`width` : the maximum image width

`height` : the maximum image height

`path` : string containing the filename of the image to load

`flags` : flags specific to the particular image loader

GrLoadImageFromFile ()

```
GR_IMAGE_ID GrLoadImageFromFile               (char *path,
                                                int flags);
```

Loads the specified image file into a newly created server image buffer and returns the ID of the buffer. The image type is automatically detected using the magic numbers in the image header (ie. the filename extension is irrelevant). The currently supported image types include GIF, JPEG, Windows BMP, PNG, XPM, and both ascii and binary variants of PBM, PGM, and PPM. However the image types supported by a particular server depend on which image types were enabled in the server configuration at build time.

`path` : string containing the filename of the image to load
`flags` : flags specific to the particular image loader
Returns : ID of the image buffer the image was loaded into

GrDrawImageToFit ()

```
void        GrDrawImageToFit                (GR_DRAW_ID id,
                                             GR_GC_ID gc,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_SIZE width,
                                             GR_SIZE height,
                                             GR_IMAGE_ID im-
ageid);
```

Draws the image from the specified image buffer at the specified position on the specified drawable using the specified graphics context. The width and height values specify the size of the image to draw- if the actual image is a different size, it will be scaled to fit.

`id` : the ID of the drawable to draw the image onto
`gc` : the ID of the graphics context to use when drawing the image
`x` : the X coordinate to draw the image at relative to the drawable
`y` : the Y coordinate to draw the image at relative to the drawable
`width` : the maximum image width
`height` : the maximum image height
`imageid` : the ID of the image buffer containing the image to display

GrDrawImageBits ()

```
void        GrDrawImageBits                (GR_DRAW_ID id,
                                            GR_GC_ID gc,
                                            GR_COORD x,
                                            GR_COORD y,
                                            GR_IMAGE_HDR *pim-
age);
```

Draws the image contained in the specified image structure onto the specified drawable at the specified coordinates using the specified graphics context.

id :	the ID of the drawable to draw the image onto
gc :	the ID of the graphics context to use when drawing the image
x :	the X coordinate to draw the image at relative to the drawable
y :	the Y coordinate to draw the image at relative to the drawable
pimage :	pointer to the image structure

GrText ()

```
void        GrText                         (GR_DRAW_ID id,
                                            GR_GC_ID gc,
                                            GR_COORD x,
                                            GR_COORD y,
                                            void *str,
                                            GR_COUNT count,
                                            int flags);
```

Draws the specified text string at the specified position on the specified drawable using the specified graphics context and flags. The default flags specify ASCII encoding and baseline alignment.

`id` : the ID of the drawable to draw the text string onto

`gc` : the ID of the graphics context to use when drawing the text string

`x` : the X coordinate to draw the string at relative to the drawable

`y` : the Y coordinate to draw the string at relative to the drawable

`str` : the text string to draw

`count` : the number of characters (not bytes) in the string

`flags` : flags specifying text encoding, alignment, etc.

events (3)

Name

`events` —

Synopsis

```
void        GrSelectEvents                (GR_WINDOW_ID wid,
                                           GR_EVENT_MASK eventmask);
void        GrGetNextEvent                (GR_EVENT *ep);
void        GrGetNextEventTimeout         (GR_EVENT *ep,
                                           GR_TIMEOUT timeout);
void        GrCheckNextEvent              (GR_EVENT *ep);
int         GrPeekEvent                   (GR_EVENT *ep);
```

Description

Details

GrSelectEvents ()

```
void        GrSelectEvents                   (GR_WINDOW_ID wid,
                                              GR_EVENT_MASK event-
mask);
```

Select the event types which should be returned for the specified window.

wid : the ID of the window to set the event mask of
eventmask : a bit field specifying the desired event mask

GrGetNextEvent ()

```
void        GrGetNextEvent                   (GR_EVENT *ep);
```

Gets the next event from the event queue and places it in the specified GR_EVENT structure. If the queue is currently empty, we sleep until the next event arrives from the server or input is read on a file descriptor previously specified by GrRegisterInput().

ep : pointer to the GR_EVENT structure to return the event in

GrGetNextEventTimeout ()

```
void        GrGetNextEventTimeout            (GR_EVENT *ep,
                                            GR_TIMEOUT time-
out);
```

Gets the next event from the event queue and places it in the specified GR_EVENT structure. If the queue is currently empty, we sleep until the next event arrives from the server, input is read on a file descriptor previously specified by GrRegisterInput(), or a timeout occurs. Note that a value of 0 for the timeout parameter doesn't mean "timeout after 0 milliseconds" but is in fact a magic number meaning "never time out".

`ep` : pointer to the GR_EVENT structure to return the event in

`timeout` : the number of milliseconds to wait before timing out

GrCheckNextEvent ()

```
void        GrCheckNextEvent               (GR_EVENT *ep);
```

Gets the next event from the event queue if there is one, or returns immediately with an event type of GR_EVENT_TYPE_NONE if it is empty.

`ep` : pointer to the GR_EVENT structure to return the event in

GrPeekEvent ()

```
int         GrPeekEvent                    (GR_EVENT *ep);
```

Fills in the specified event structure with a copy of the next event on the queue, without actually removing it from the queue. An event type of GR_EVENT_TYPE_NONE is given if the queue is empty.

`ep` : pointer to the GR_EVENT structure to return the event in

Returns : 1 if an event was returned, or 0 if the queue was empty

fonts (3)

Name

`fonts` —

Synopsis

```
GR_FONT_ID  GrCreateFont                      (GR_CHAR *name,
                                               GR_COORD height,
                                               GR_LOGFONT *plogfont);
void        GrSetFontSize                     (GR_FONT_ID fontid,
                                               GR_COORD size);
void        GrSetFontRotation                 (GR_FONT_ID fontid,
                                               int tenthsdegrees);
void        GrSetFontAttr                     (GR_FONT_ID fontid,
                                               int setflags,
                                               int clrflags);
void        GrDestroyFont                     (GR_FONT_ID fontid);
void        GrGetFontInfo                     (GR_FONT_ID font,
                                               GR_FONT_INFO *fip);
```

Description

Details

GrCreateFont ()

```
GR_FONT_ID  GrCreateFont                       (GR_CHAR *name,
                                                GR_COORD height,
                                                GR_LOGFONT *plog-
font);
```

Attempts to locate a font with the desired attributes and returns a font ID number which can be used to refer to it. If the plogfont argument is not NULL, the values in that structure will be used to choose a font. Otherwise, if the height is non zero, the built in font with the closest height to that specified will be used. If the height is zero, the built in font with the specified name will be used. If the desired font is not found, the first built in font will be returned as a last resort.

`name` :	string containing the name of a built in font to look for
`height` :	the desired height of the font
`plogfont` :	pointer to a LOGFONT structure
Returns :	a font ID number which can be used to refer to the font

GrSetFontSize ()

```
void        GrSetFontSize                      (GR_FONT_ID fontid,
                                                GR_COORD size);
```

Changes the size of the specified font to the specified size.

fontid : the ID number of the font to change the size of
size :

GrSetFontRotation ()

```
void        GrSetFontRotation                (GR_FONT_ID fontid,
                                              int tenthsdegrees);
```

Changes the rotation of the specified font to the specified angle.

fontid : the ID number of the font to rotate
tenthsdegrees :

GrSetFontAttr ()

```
void        GrSetFontAttr                    (GR_FONT_ID fontid,
                                              int setflags,
                                              int clrflags);
```

Changes the attributes (GR_TFKERNING, GR_TFANTIALIAS, GR_TFUNDERLINE, etc.) of the specified font according to the set and clear mask arguments.

fontid : the ID of the font to set the attributes of
setflags : mask specifying attribute flags to set
clrflags : mask specifying attribute flags to clear

GrDestroyFont ()

```
void        GrDestroyFont                  (GR_FONT_ID fontid);
```

Frees all resources associated with the specified font ID, and if the font is a non built in type and this is the last ID referring to it, unloads the font from memory.

`fontid` : the ID of the font to destroy

GrGetFontInfo ()

```
void        GrGetFontInfo                  (GR_FONT_ID font,
                                            GR_FONT_INFO *fip);
```

Fills in the specified GR_FONT_INFO structure with information regarding the specified font.

`font` :

`fip` : pointer to a GR_FONT_INFO structure

pointer (3)

Name

`pointer` —

Synopsis

```
void        GrSetCursor                     (GR_WINDOW_ID wid,
                                             GR_SIZE width,
                                             GR_SIZE height,
                                             GR_COORD hotx,
                                             GR_COORD hoty,
                                             GR_COLOR foreground,
                                             GR_COLOR background,
                                             GR_BITMAP *fbbitmap,
                                             GR_BITMAP *bgbitmap,
                                             int flags);
void        GrMoveCursor                    (GR_COORD x,
                                             GR_COORD y);
void        GrInjectPointerEvent            (MWCOORD x,
                                             MWCOORD y,
                                             int button,
                                             int visible);
```

Description

Details

GrSetCursor ()

```
void        GrSetCursor                     (GR_WINDOW_ID wid,
                                             GR_SIZE width,
                                             GR_SIZE height,
                                             GR_COORD hotx,
                                             GR_COORD hoty,
                                             GR_COLOR foreground,
                                             GR_COLOR background,
                                             GR_BITMAP *fbbitmap,
                                             GR_BITMAP *bgbitmap,
                                             int flags);
```

Specifies a cursor (mouse pointer graphic) to display when the mouse pointer is over the specified window and subsequently created children. Points in the bitmap which have neither the foreground or background bits set are not painted. If the flags argument is set to GR_CURSOR_HIDE or GR_CURSOR_SHOW, the mouse pointer will be made invisible or returned to visiblity respectively. When using the GR_CURSOR_HIDE and GR_CURSOR_SHOW flags, all other arguments are ignored.

`wid` :	the ID of the window to set the cursor of
`width` :	the width of the pointer bitmap
`height` :	the height of the pointer bitmap
`hotx` :	the X coordinate within the bitmap used as the target of the pointer
`hoty` :	the Y coordinate within the bitmap used as the target of the pointer
`foreground` :	the colour to use for the foreground of the pointer
`background` :	the colour to use for the background of the pointer
`fbbitmap` :	
`bgbitmap` :	pointer to bitmap data specifying the background of the pointer
`flags` :	extra flags controlling visibility of pointer, etc.

GrMoveCursor ()

```
void         GrMoveCursor                    (GR_COORD x,
                                              GR_COORD y);
```

Moves the cursor (mouse pointer) to the specified coordinates. The coordinates are relative to the root window, where (0,0) is the upper left hand corner of the screen. The reference point used for the pointer is that of the "hot spot". After moving the pointer, the graphic used for the pointer will change to the graphic defined for use in the window which it is over.

`x` :	the X coordinate to move the pointer to
`y` :	the Y coordinate to move the pointer to

GrInjectPointerEvent ()

```
void        GrInjectPointerEvent            (MWCOORD x,
                                             MWCOORD y,
                                             int button,
                                             int visible);
```

Sets the pointer invisible if the visible parameter is GR_FALSE, or visible if it is GR_TRUE, then moves the pointer to the specified position and generates a mouse event with the specified button status. Also performs a GrFlush() so that the event takes effect immediately.

`x` :	the X coordinate of the pointer event relevant to the root window
`y` :	the Y coordinate of the pointer event relevant to the root window
`button` :	the pointer button status
`visible` :	whether to display the pointer after the event

colours (3)

Name

`colours` —

Synopsis

```
void        GrGetSystemPalette               (GR_PALETTE *pal);
void        GrSetSystemPalette               (GR_COUNT first,
                                              GR_PALETTE *pal);
void        GrFindColor                      (GR_COLOR c,
                                              GR_PIXELVAL *retpixel);
GR_COLOR    GrGetSysColor                    (int index);
```

Description

Details

GrGetSystemPalette ()

```
void        GrGetSystemPalette               (GR_PALETTE *pal);
```

Retrieves the system palette and places it in the specified palette structure.

pal :	pointer to a palette structure to fill in with the system palette

GrSetSystemPalette ()

```
void        GrSetSystemPalette                (GR_COUNT first,
                                               GR_PALETTE *pal);
```

Sets the system palette to the values stored in the specified palette structure. The values before the specified first value are not set.

first :	the first palette value to set
pal :	pointer to a palette structure containing the new values

GrFindColor ()

```
void        GrFindColor                       (GR_COLOR c,
                                               GR_PIXELVAL *ret-
pixel);
```

Calculates the pixel value to use to display the specified colour value. The colour value is specified as a GR_COLOR, which is a 32 bit truecolour value stored as RGBX. The pixel value size depends on the architecture.

c :	the colour value to find
retpixel :	pointer to the returned pixel value

GrGetSysColor ()

```
GR_COLOR    GrGetSysColor                     (int index);
```

`index` : an index into the server’s colour look up table
Returns : the colour found at the specified index

regions (3)

Name

`regions` —

Synopsis

```
GR_REGION_ID GrNewRegion                    (void);
void        GrDestroyRe-
gion                 (GR_REGION_ID region);
void        GrUnionRectWithRe-
gion           (GR_REGION_ID region,
                                             GR_RECT *rect);
void        GrUnionRe-
gion                   (GR_REGION_ID dst_rgn,
                                             GR_REGION_ID src_rgn1,
                                             GR_REGION_ID src_rgn2);
void        GrSubtractRe-
gion                (GR_REGION_ID dst_rgn,
                                             GR_REGION_ID src_rgn1,
                                             GR_REGION_ID src_rgn2);
void        GrXorRe-
gion                     (GR_REGION_ID dst_rgn,
                                             GR_REGION_ID src_rgn1,
                                             GR_REGION_ID src_rgn2);
void         GrIntersectRe-
gion               (GR_REGION_ID dst_rgn,
                                             GR_REGION_ID src_rgn1,
                                             GR_REGION_ID src_rgn2);
void         GrSetGCRegion                  (GR_GC_ID gc,
                                             GR_REGION_ID region);
GR_BOOL      GrPointInRe-
gion                 (GR_REGION_ID region,
                                             GR_COORD x,
                                             GR_COORD y);
int          GrRectInRe-
gion                  (GR_REGION_ID region,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_COORD w,
                                             GR_COORD h);
GR_BOOL      GrEmptyRe-
gion                   (GR_REGION_ID region);
GR_BOOL      GrEqualRegion                  (GR_REGION_ID rgn1,
                                             GR_REGION_ID rgn2);
void         GrOffsetRe-
gion                  (GR_REGION_ID region,
                                             GR_SIZE dx,
                                             GR_SIZE dy);
int          GrGetRegion-
Box                  (GR_REGION_ID region,
                                             GR_RECT *rect);
GR_REGION_ID GrNewPolygonRegion             (int mode,
                                             GR_COUNT count,
                                             GR_POINT *points);
```

Description

Details

GrNewRegion ()

```
GR_REGION_ID GrNewRegion                    (void);
```

Creates a new region structure and returns the ID used to refer to it. The structure is initialised with a set of default parameters.

Returns : the ID of the newly created region

GrDestroyRegion ()

```
void        GrDestroyRegion                 (GR_REGION_ID re-
gion);
```

Destroys the region structure with the specified ID.

`region` : the ID of the region structure to destroy

GrUnionRectWithRegion ()

```
void        GrUnionRectWithRegion           (GR_REGION_ID re-
gion,
                                              GR_RECT *rect);
```

Makes a union of the specified region and the specified rectangle and places the result back in the source region.

`region` :	the ID of the region to modify
`rect` :	a pointer to the rectangle to add to the region

GrUnionRegion ()

```
void        GrUnionRe-
gion                  (GR_REGION_ID dst_rgn,
                                             GR_REGION_ID src_rgn1,
                                             GR_REGION_ID src_rgn2);
```

Makes a union of the specified source regions and places the result in the specified destination region.

`dst_rgn` :	the ID of the destination region
`src_rgn1` :	the ID of the first source region
`src_rgn2` :	the ID of the second source region

GrSubtractRegion ()

```
void        GrSubtractRe-
gion                (GR_REGION_ID dst_rgn,
                                             GR_REGION_ID src_rgn1,
                                             GR_REGION_ID src_rgn2);
```

Subtracts the second source region from the first source region and places the result in the specified destination region.

`dst_rgn` :	the ID of the destination region
`src_rgn1` :	the ID of the first source region
`src_rgn2` :	the ID of the second source region

GrXorRegion ()

```
void        GrXorRe-
gion                     (GR_REGION_ID dst_rgn,
                                             GR_REGION_ID src_rgn1,
                                             GR_REGION_ID src_rgn2);
```

Performs a logical exclusive OR operation on the specified source regions and places the result in the destination region. The destination region will contain only the parts of the source regions which do not overlap.

`dst_rgn` : the ID of the destination region

`src_rgn1` : the ID of the first source region

`src_rgn2` : the ID of the second source region

GrIntersectRegion ()

```
void        GrIntersectRe-
gion              (GR_REGION_ID dst_rgn,
                                             GR_REGION_ID src_rgn1,
                                             GR_REGION_ID src_rgn2);
```

Calculates the intersection of the two specified source regions and places the result in the specified destination region. The destination region will contain only the parts of the source regions which overlap each other.

`dst_rgn` : the ID of the destination region

`src_rgn1` : the ID of the first source region

`src_rgn2` : the ID of the second source region

GrSetGCRegion ()

```
void        GrSetGCRegion                   (GR_GC_ID gc,
                                              GR_REGION_ID re-
gion);
```

Sets the clip mask of the specified graphics context to the specified region. Subsequent drawing operations using this graphics context will not draw outside the specified region. The region ID can be set to 0 to remove the clipping region from the specified graphics context.

`gc` :	the ID of the graphics context to set the clip mask of
`region` :	the ID of the region to use as the clip mask

GrPointInRegion ()

```
GR_BOOL     GrPointInRegion                 (GR_REGION_ID re-
gion,
                                              GR_COORD x,
                                              GR_COORD y);
```

Tests whether the specified point is within the specified region, and then returns either True or False depending on the result.

`region` :	the ID of the region to examine
`x` :	the X coordinate of the point to test for
`y` :	the Y coordinate of the point to test for
Returns :	True if the point is within the region, or False otherwise

GrRectInRegion ()

```
int         GrRectInRegion                  (GR_REGION_ID region,
                                             GR_COORD x,
                                             GR_COORD y,
                                             GR_COORD w,
                                             GR_COORD h);
```

Tests whether the specified rectangle is contained within the specified region. Returns GR_RECT_OUT if it is not inside it at all, GR_RECT_ALLIN if it is completely contained within the region, or GR_RECT_PARTIN if it is partially contained within the region.

`region` : the ID of the region to examine
`x` : the X coordinates of the rectangle to test
`y` : the Y coordinates of the rectangle to test
`w` : the width of the rectangle to test
`h` : the height of the rectangle to test
Returns : GR_RECT_PARTIN, GR_RECT_ALLIN, or GR_RECT_OUT

GrEmptyRegion ()

```
GR_BOOL     GrEmptyRegion                   (GR_REGION_ID region);
```

Determines whether the specified region is empty, and returns GR_TRUE if it is, or GR_FALSE otherwise.

`region` : the ID of the region to examine
Returns : GR_TRUE if the region is empty, or GR_FALSE if it is not

GrEqualRegion ()

```
GR_BOOL     GrEqualRegion                  (GR_REGION_ID rgn1,
                                            GR_REGION_ID rgn2);
```

Determines whether the specified regions are identical, and returns GR_TRUE if it is, or GR_FALSE otherwise.

`rgn1` : the ID of the first region to examine

`rgn2` : the ID of the second region to examine

Returns : GR_TRUE if the regions are equal, or GR_FALSE otherwise

GrOffsetRegion ()

```
void        GrOffsetRegion                 (GR_REGION_ID re-
gion,

                                            GR_SIZE dx,
                                            GR_SIZE dy);
```

Offsets the specified region by the specified distance.

`region` : the ID of the region to offset

`dx` : the distance to offset the region by in the X axis

`dy` : the distance to offset the region by in the Y axis

GrGetRegionBox ()

```
int         GrGetRegionBox                 (GR_REGION_ID re-
gion,

                                            GR_RECT *rect);
```

Fills in the specified rectangle structure with a bounding box that would completely enclose the specified region, and also returns the type of the specified region.

`region` :	the ID of the region to get the bounding box of
`rect` :	pointer to a rectangle structure
Returns :	the region type

GrNewPolygonRegion ()

```
GR_REGION_ID GrNewPolygonRegion             (int mode,
                                             GR_COUNT count,
                                             GR_POINT *points);
```

Creates a new region structure, fills it with the region described by the specified polygon, and returns the ID used to refer to it.

`mode` :	the polygon mode to use (GR_POLY_EVENODD or GR_POLY_WINDING)
`count` :	the number of points in the polygon
`points` :	pointer to an array of point structures describing the polygon
Returns :	the ID of the newly allocated region structure, or 0 on error

selections (3)

Name

`selections` —

Synopsis

```
void          GrSetSelectionOwner             (GR_WINDOW_ID wid,
                                               GR_CHAR *typelist);
GR_WINDOW_ID GrGetSelec-
tionOwner            (GR_CHAR **typelist);
void          GrRequestClientData             (GR_WINDOW_ID wid,
                                               GR_WINDOW_ID rid,
                                               GR_SERIALNO serial,
                                               GR_MIMETYPE mimetype);
void          GrSendClientData                (GR_WINDOW_ID wid,
                                               GR_WINDOW_ID did,
                                               GR_SERIALNO serial,
                                               GR_LENGTH len,
                                               void *data);
```

Description

Details

GrSetSelectionOwner ()

```
void        GrSetSelectionOwner             (GR_WINDOW_ID wid,
                                              GR_CHAR *typelist);
```

Sets the current selection (otherwise known as the clipboard) ownership to the specified window. Specifying an owner of 0 disowns the selection. The typelist argument is a list of mime types (seperated by space characters) which the window is able to supply the data as. At least one type must be specified unless you are disowning the selection (typically text/plain for plain ASCII text or text/uri-list for a filename).

The window which owns the current selection must be prepared to handle SELECTION_LOST events (received when another window takes ownership of the selection) and CLIENT_DATA_REQ events (received when a client wishes to retreive the selection data).

`wid` :	the ID of the window to set the selection owner to
`typelist` :	list of mime types selection data can be supplied as

GrGetSelectionOwner ()

```
GR_WINDOW_ID GrGetSelectionOwner            (GR_CHAR **type-
list);
```

Finds the window which currently owns the selection and returns its ID, or 0 if no window currently owns the selection. A pointer to the list of mime types the selection owner is capable of supplying is placed in the pointer specified by the typelist argument. The typelist is null terminated, and the fields are seperated by space characters. It is the callers responsibility to free the typelist string, as it is allocated dynamically. If the allocation fails, it will be set to a NULL pointer, so remember to check the value of it before using it.

`typelist` :	pointer used to return the list of available mime types
Returns :	the ID of the window which currently owns the selection, or 0

GrRequestClientData ()

```
void        GrRequestClientData             (GR_WINDOW_ID wid,
                                             GR_WINDOW_ID rid,
                                             GR_SERIALNO serial,
                                             GR_MIMETYPE mime-
type);
```

Sends a CLIENT_DATA_REQ event to the specified window. Used for requesting both selection and "drag and drop" data. The mimetype argument specifies the format of the data you would like to receive, as an index into the list returned by GrGetSelectionOwner (the first type in the list is index 0). The server makes no guarantees as to when, or even if, the client will reply to the request. If the client does reply, the reply will take the form of one or more CLIENT_DATA events. The request serial number is typically a unique ID which the client can assign to a request in order for it to be able to keep track of transfers (CLIENT_DATA events contain the same number in the sid field). Remember to free the data field of the CLIENT_DATA events as they are dynamically allocated. Also note that if the allocation fails the data field will be set to NULL, so you should check the value before using it.

`wid` :	the ID of the window requesting the data
`rid` :	the ID of the window to request the data from
`serial` :	the serial number of the request
`mimetype` :	the number of the desired mime type to request

GrSendClientData ()

```
void        GrSendClientData                (GR_WINDOW_ID wid,
                                             GR_WINDOW_ID did,
                                             GR_SERIALNO serial,
                                             GR_LENGTH len,
                                             void *data);
```

Used as the response to a CLIENT_DATA_REQ event. Sends the specified data of the specified length to the specified window using the specified source window ID and transfer serial number. Any fragmentation of the data into multiple CLIENT_DATA events which is required is handled automatically. The serial number should always be set to the value supplied by the CLIENT_DATA_REQ event.

`wid` :	the ID of the window sending the data
`did` :	the ID of the destination window
`serial` :	
`len` :	the number of bytes of data to transfer
`data` :	pointer to the data to transfer

misc (3)

Name

`misc` —

Synopsis

```
void        GrReqShmCmds                    (long shmsize);
void        GrInjectKeyboardEvent           (GR_WINDOW_ID wid,
                                             GR_UNICODE uch,
                                             GR_CHAR ch,
                                             int modif,
                                             int special,
                                             un-
signed char content);
void        GrRegisterInput                 (int fd);
void        GrPrepareSelect                 (int *maxfd,
                                             void *rfdset);
void        GrServiceSelect                 (void *rfdset,
                                             GR_FNCALLBACKEVENT fncb);
void        GrBell                          (void);
void        GrSetScreenSaverTimeout         (GR_TIMEOUT time-
out);
```

Description

Details

GrReqShmCmds ()

```
void        GrReqShmCmds                    (long shmsize);
```

Requests a shared memory area of the specified size to use for transferring command arguments. This is faster but less portable than the standard BSD sockets method of communication (and of course will only work if the client and server are on the same machine). Apart from the initial allocation of the area using this call, the use of shared memory is completely transparent. Additionally, if the allocation fails we silently and automatically fall back on socket communication. It is safe to call this function even if shared memory support is not compiled in; it will simply do nothing.

FIXME: how does the user decide what size of shared memory area to allocate?

`shmsize` : the size of the shared memory area to allocate

GrInjectKeyboardEvent ()

```
void        GrInjectKeyboardEvent           (GR_WINDOW_ID wid,
                                              GR_UNICODE uch,
                                              GR_CHAR ch,
                                              int modif,
                                              int special,
                                              unsigned char con-
tent);
```

Sends a keyboard event to the specified window, or to the window with the current keyboard focus if 0 is used as the ID. The other arguments correspond directly to the fields of the same names in the keyboard event structure.

`wid` : ID of the window to send the event to, or 0

`uch` : 32 bit Unicode keystroke value to inject

`ch` : 8 bit ascii keystroke value to inject

`modif` :

`special` : special keys to inject

`content` : mask specifying which arguments are valid

GrRegisterInput ()

```
void        GrRegisterInput                 (int fd);
```

Register an extra file descriptor to monitor in the main select() call. An event will be returned when the fd has data waiting to be read if that event has been selected for.

fd :	the file descriptor to monitor

GrPrepareSelect ()

```
void        GrPrepareSelect                 (int *maxfd,
                                             void *rfdset);
```

Prepare for a GrServiceSelect function by asking the server to send the next event but not waiting around for it to arrive and initialising the specified fd_set structure with the client/server socket descriptor and any previously registered external file descriptors. Also compares the current contents of maxfd, the client/server socket descriptor, and the previously registered external file descriptors, and returns the highest of them in maxfd.

maxfd :	pointer to a variable which the highest in use fd will be written to
rfdset :	pointer to the file descriptor set structure to use

GrServiceSelect ()

```
void        GrServiceSelect                 (void *rfdset,
                                             GR_FNCALLBACKEVENT fncb);
```

Used by GrMainLoop() to call the specified callback function when an event arrives or there is data waiting on an external fd specified by GrRegisterInput().

`rfdset` : pointer to the file descriptor set to monitor
`fncb` : pointer to the function to call when an event needs handling

GrBell ()

```
void        GrBell                          (void);
```

Asks the server to ring the console bell on behalf of the client (intended for terminal apps to be able to ring the bell on the server even if they are running remotely).

GrSetScreenSaverTimeout ()

```
void        GrSetScreenSaverTimeout         (GR_TIMEOUT time-
out);
```

Sets the number of seconds of inactivity before a screen saver activate event is sent to the root window ID. A value of 0 activates the screen saver immediately, and a value of -1 disables the screen saver function.

`timeout` : the number of seconds of inactivity before screen saver activates