# Microwindows Nano-X API Reference Manual

Generated by Doxygen 1.2.18

Tue Aug 24 19:46:33 2004

# Contents

# Chapter 1

# Microwindows Nano-X API Module Index

## 1.1 Microwindows Nano-X API Modules

Here is a list of all modules:

# Chapter 2

# Microwindows Nano-X API Data Structure Index

## 2.1 Microwindows Nano-X API Data Structures

Here are the data structures with brief descriptions:

# Chapter 3

# Microwindows Nano-X API Page Index

## 3.1 Microwindows Nano-X API Related Pages

Here is a list of all related documentation pages:

# Chapter 4

# Microwindows Nano-X API Module Documentation

## 4.1 Nano-X public API

This is the API which Nano-X applications use.

### Modules

- Nano-X color/palette management API.

  *Functions for querying and modifying the palette on palette-based Nano-X systems.*

- Nano-X cursor API.

  *Functions for controlling the appearance of the mouse pointer.*

- Nano-X drawing API.

  *Functions for actually drawing primitive shapes on the screen.*

- Nano-X events API.

  *The Nano-X event mechanism.*

- Nano-X font API.

  *Functions for handling fonts and drawing text.*

- Nano-X basic API.

  *Functions to initialise and close Nano-X.*

- Nano-X image file API.

  *Functions to draw images from standard image file formats.*

- Nano-X miscellaneous APIs.

*Functions that didn't fit anywhere else.*

- Nano-X region API.

  *Functions for handling clipping regions - these are used for clipping drawing, and for non-rectangular windows.*

- Nano-X clipboard API.

  *Functions for handling the current selection on the clipboard.*

- Nano-X timer API.

  *Functions for handling timers and delays.*

- Nano-X window API.

  *Functions for handling windows on the screen.*

### 4.1.1 Detailed Description

This is the API which Nano-X applications use.

## 4.2 Nano-X color/palette management API.

Functions for querying and modifying the palette on palette-based Nano-X systems.

### Functions

- GR_COLOR GrGetSysColor (int index)

  *Returns the colour at the specified index into the server's color look up table.*

- void GrGetSystemPalette (GR_PALETTE ∗pal)

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

- void GrSetSystemPalette (GR_COUNT first, GR_PALETTE ∗pal)

  *Sets the system palette to the values stored in the specified palette structure.*

- void GrFindColor (GR_COLOR c, GR_PIXELVAL ∗retpixel)

  *Calculates the pixel value to use to display the specified colour value.*

### 4.2.1 Detailed Description

Functions for querying and modifying the palette on palette-based Nano-X systems.

### 4.2.2 Function Documentation

#### 4.2.2.1 void GrFindColor (GR_COLOR *c*, GR_PIXELVAL ∗ *retpixel*)

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.

**Parameters:**

***c*** the colour value to find

***retpixel*** pointer to the returned pixel value

#### 4.2.2.2 GR_COLOR GrGetSysColor (int *index*)

Returns the colour at the specified index into the server's color look up table.

The colours in the table are those with names like ”GR_COLOR_DESKTOP”, ”GR_-COLOR_ACTIVECAPTION”, ”GR_COLOR_APPWINDOW”, etc. as listed in nano-X.h

**Parameters:**

***index*** An index into the server's colour look up table.

**Returns:**

The color found at the specified index.

#### 4.2.2.3 void GrGetSystemPalette (GR_PALETTE ∗ *pal*)

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

**Parameters:**

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

#### 4.2.2.4 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.

**Parameters:**

***first*** the first palette value to set

***pal*** pointer to a palette structure containing the new values

## 4.3 Nano-X cursor API.

Functions for controlling the appearance of the mouse pointer.

### Functions

- void GrSetWindowCursor (GR_WINDOW_ID wid, GR_CURSOR_ID cid)

  *Specify a cursor for a window.*

- GR_CURSOR_ID GrNewCursor (GR_SIZE width, GR_SIZE height, GR_COORD hotx, GR_COORD hoty, GR_COLOR foreground, GR_COLOR background, GR_BITMAP *fgbitmap, GR_BITMAP *bgbitmap)

  *Creates a server-based cursor (mouse graphic) resource.*

- void GrMoveCursor (GR_COORD x, GR_COORD y)

  *Moves the cursor (mouse pointer) to the specified coordinates.*

- void GrDestroyCursor (GR_CURSOR_ID cid)

  *Destroys the specified server-based cursor and reclaims the memory used by it.*

### 4.3.1 Detailed Description

Functions for controlling the appearance of the mouse pointer.

### 4.3.2 Function Documentation

#### 4.3.2.1 void GrDestroyCursor (GR_CURSOR_ID *cid*)

Destroys the specified server-based cursor and reclaims the memory used by it.

**Parameters:**

***cid*** ID of the cursor to destory

#### 4.3.2.2 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.

**Parameters:**

***x*** the X coordinate to move the pointer to

***y*** the Y coordinate to move the pointer to

#### 4.3.2.3 GR_CURSOR_ID GrNewCursor (GR_SIZE *width*, GR_SIZE *height*, GR_COORD *hotx*, GR_COORD *hoty*, GR_COLOR *foreground*, GR_COLOR *background*, GR_BITMAP ∗ *fgbitmap*, GR_BITMAP ∗ *bgbitmap*)

Creates a server-based cursor (mouse graphic) resource.

**Parameters:**

***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

***fgbitmap*** pointer to bitmap data specifying the foreground of the pointer

***bgbitmap*** pointer to bitmap data specifying the background of the pointer

#### 4.3.2.4 void GrSetWindowCursor (GR_WINDOW_ID *wid*, GR_CURSOR_ID *cid*)

Specify a cursor for a window.

This cursor will only be used within that window, and by default for its new children. If the cursor is currently within this window, it will be changed to the new one immediately. If the new cursor ID is 0, revert to the root window cursor.

**Parameters:**

***wid*** the ID of the window to set the cursor for

***cid*** the cursor ID

## 4.4 Nano-X drawing API.

Functions for actually drawing primitive shapes on the screen.

### Functions

- 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.*

- GR_GC_ID GrNewGC (void)

  *Creates a new graphics context structure.*

- GR_GC_ID GrCopyGC (GR_GC_ID gc)

  *Creates a new graphics context structure and copies the settings from an already existing graphics context.*

- void GrDestroyGC (GR_GC_ID gc)

  *Destroys a graphics context structure.*

- void GrSetGCClipOrigin (GR_GC_ID gc, int x, int y)

  *Sets the X,Y origin of the user clip region in the specified graphics context.*

- void GrSetGCGraphicsExposure (GR_GC_ID gc, GR_BOOL exposure)

  *Controls if GR_EVENT_TYPE_EXPOSURE events are sent as a result of GrCopyArea using the specified graphics context.*

- void GrClearArea (GR_WINDOW_ID wid, GR_COORD x, GR_COORD y, GR_SIZE width, GR_SIZE height, GR_BOOL exposeflag)

  *Clears the specified window by to its background color or pixmap.*

- void GrSetGCForeground (GR_GC_ID gc, GR_COLOR foreground)

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

- void GrSetGCBackground (GR_GC_ID gc, GR_COLOR background)

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

- void GrSetGCForegroundPixelVal (GR_GC_ID gc, GR_PIXELVAL foreground)

  *Changes the foreground colour of the specified graphics context to the specified hardware pixel value.*

- void GrSetGCBackgroundPixelVal (GR_GC_ID gc, GR_PIXELVAL background)

*Changes the background colour of the specified graphics context to the specified hardware pixel value.*

- 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.*

- void GrSetGCLineAttributes (GR_GC_ID, int)

  *Changes the line style to either SOLID or ON OFF DASHED.*

- void GrSetGCDash (GR_GC_ID, char ∗, int)

  *FIXME.*

- void GrSetGCFillMode (GR_GC_ID, int)

  *FIXME.*

- void GrSetGCStipple (GR_GC_ID gc, GR_BITMAP ∗bitmap, int width, int height)

  *FIXME.*

- void GrSetGCTile (GR_GC_ID gc, GR_WINDOW_ID pixmap, int width, int height)

  *FIXME.*

- void GrSetGCTSOffset (GR_GC_ID gc, int xoff, int yoff)

  *FIXME.*

- 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.*

- 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.*

- 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.*

- 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.*

- 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.*

- 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.*

- 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.*

- 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.*

- 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 imagebits argument at the specified position on the specified drawable using the specified graphics context.*

- void GrDrawImageBits (GR_DRAW_ID id, GR_GC_ID gc, GR_COORD x, GR_COORD y, GR_IMAGE_HDR ∗pimage)

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

- 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.*

- 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, unsigned long 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.*

- void GrReadArea (GR_DRAW_ID id, GR_COORD x, GR_COORD y, GR_SIZE width, GR_SIZE height, GR_PIXELVAL ∗pixels)

  *Reads the pixel data of the specified size from the specified position on the specified drawable into the specified pixel array.*

- 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.*

- void GrPoints (GR_DRAW_ID id, GR_GC_ID gc, GR_COUNT count, GR_POINT ∗pointtable)

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

- void GrPoly (GR_DRAW_ID id, GR_GC_ID gc, GR_COUNT count, GR_POINT ∗pointtable)

  *Draws an unfilled polygon on the specified drawable using the specified graphics context.*

- void GrFillPoly (GR_DRAW_ID id, GR_GC_ID gc, GR_COUNT count, GR_POINT ∗pointtable)

  *Draws a filled polygon on the specified drawable using the specified graphics context.*

- void GrStretchArea (GR_DRAW_ID dstid, GR_GC_ID gc, GR_COORD dx1, GR_COORD dy1, GR_COORD dx2, GR_COORD dy2, GR_DRAW_ID srcid, GR_COORD sx1, GR_COORD sy1, GR_COORD sx2, GR_COORD sy2, unsigned long op)

  *Copies a region from one drawable to another.*

### 4.4.1 Detailed Description

Functions for actually drawing primitive shapes on the screen.

### 4.4.2 Function Documentation

#### 4.4.2.1 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.

**Parameters:**

***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

#### 4.4.2.2 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.

**Parameters:**

***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

#### 4.4.2.3 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.

**Parameters:**

***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

#### 4.4.2.4 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 imagebits 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.

**Parameters:**

***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*** pointer to the bitmap data

#### 4.4.2.5 void GrClearArea (GR_WINDOW_ID *wid*, GR_COORD *x*, GR_COORD *y*, GR_SIZE *width*, GR_SIZE *height*, GR_BOOL *exposeflag*)

Clears the specified window by to its background color or pixmap.

If exposeflag is non zero, an exposure event is generated for the window after it has been cleared.

**Parameters:**

***wid*** Window ID.

***x*** X co-ordinate of rectangle to clear.

***y*** Y co-ordinate of rectangle to clear.

***width*** Width of rectangle to clear.

***height*** Height of rectangle to clear.

***exposeflag*** A flag indicating whether to also generate an exposure event.

#### 4.4.2.6 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*, unsigned long *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.

**Parameters:**

***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

#### 4.4.2.7 GR_GC_ID GrCopyGC (GR_GC_ID *gc*)

Creates a new graphics context structure and copies the settings from an already existing graphics context.

**Parameters:**

***gc*** The already existing graphics context to copy the parameters from.

**Returns:**

The ID of the newly created graphics context or 0 on error.

#### 4.4.2.8 void GrDestroyGC (GR_GC_ID *gc*)

Destroys a graphics context structure.

**Parameters:**

***gc*** the ID of the graphics context structure to destroy

#### 4.4.2.9 void GrDrawImageBits (GR_DRAW_ID *id*, GR_GC_ID *gc*, GR_COORD *x*, GR_COORD *y*, GR_IMAGE_HDR ∗ *pimage*)

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

**Parameters:**

***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

#### 4.4.2.10 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.

**Parameters:**

***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

#### 4.4.2.11 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.

**Parameters:**

***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

#### 4.4.2.12 void GrFillPoly (GR_DRAW_ID *id*, GR_GC_ID *gc*, GR_COUNT *count*, GR_POINT ∗ *pointtable*)

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.

**Parameters:**

***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

#### 4.4.2.13 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.

**Parameters:**

***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

#### 4.4.2.14 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.

**Parameters:**

***gc*** A graphics context.

***gcip*** Pointer to a GR_GC_INFO structure to store the result.

#### 4.4.2.15 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.

**Parameters:**

*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

#### 4.4.2.16 GR_GC_ID GrNewGC (void)

Creates a new graphics context structure.

The structure is initialised with a set of default parameters.

**Returns:**

The ID of the newly created graphics context or 0 on error.

#### 4.4.2.17 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.

**Parameters:**

*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

#### 4.4.2.18 void GrPoints (GR_DRAW_ID *id*, GR_GC_ID *gc*, GR_COUNT *count*, GR_POINT ∗ *pointtable*)

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

**Parameters:**

*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

#### 4.4.2.19 void GrPoly (GR_DRAW_ID *id*, GR_GC_ID *gc*, GR_COUNT *count*, GR_POINT ∗ *pointtable*)

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.

**Parameters:**

***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

#### 4.4.2.20 void GrReadArea (GR_DRAW_ID *id*, GR_COORD *x*, GR_COORD *y*, GR_SIZE *width*, GR_SIZE *height*, GR_PIXELVAL ∗ *pixels*)

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.

**Parameters:**

***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

#### 4.4.2.21 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.

**Parameters:**

***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

#### 4.4.2.22 void GrSetGCBackground (GR_GC_ID *gc*, GR_COLOR *background*)

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

**Parameters:**

***gc*** the ID of the graphics context to set the background colour of

***background*** the RGB colour to use as the new background colour

#### 4.4.2.23 void GrSetGCBackgroundPixelVal (GR_GC_ID *gc*, GR_PIXELVAL *background*)

Changes the background colour of the specified graphics context to the specified hardware pixel value.

**Parameters:**

***gc*** the ID of the graphics context to set the background colour of

***background*** the GR_PIXELVAL (i.e. hardware pixel value) to use as the new background colour

#### 4.4.2.24 void GrSetGCClipOrigin (GR_GC_ID *gc*, int *xoff*, int *yoff*)

Sets the X,Y origin of the user clip region in the specified graphics context.

**Parameters:**

***gc*** The ID of the graphics context with user clip region.

***xoff*** New X offset of user clip region.

***yoff*** New Y offset of user clip region.

#### 4.4.2.25 void GrSetGCDash (GR_GC_ID *gc*, char ∗ *dashes*, int *count*)

FIXME.

**Parameters:**

***gc*** Graphics context ID.

***dashes*** FIXME

***count*** FIXME

**Todo:**

FIXME document this

#### 4.4.2.26 void GrSetGCFillMode (GR_GC_ID *gc*, int *fillmode*)

FIXME.

**Parameters:**

***gc*** FIXME

***fillmode*** FIXME

**Todo:**

FIXME document this

#### 4.4.2.27 void GrSetGCForeground (GR_GC_ID *gc*, GR_COLOR *foreground*)

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

**Parameters:**

***gc*** the ID of the graphics context to set the foreground colour of

***foreground*** the RGB colour to use as the new foreground colour

#### 4.4.2.28 void GrSetGCForegroundPixelVal (GR_GC_ID *gc*, GR_PIXELVAL *foreground*)

Changes the foreground colour of the specified graphics context to the specified hardware pixel value.

**Parameters:**

***gc*** The ID of the graphics context to set the foreground colour of.

***foreground*** The GR_PIXELVAL (i.e. hardware pixel value) to use as the new foreground colour.

#### 4.4.2.29 void GrSetGCGraphicsExposure (GR_GC_ID *gc*, GR_BOOL *exposure*)

Controls if GR_EVENT_TYPE_EXPOSURE events are sent as a result of GrCopyArea using the specified graphics context.

**Parameters:**

***gc*** The ID of the graphics context

***exposure*** TRUE to send events, FALSE otherwise.

#### 4.4.2.30 void GrSetGCLineAttributes (GR_GC_ID *gc*, int *linestyle*)

Changes the line style to either SOLID or ON OFF DASHED.

**Parameters:**

***gc*** the ID of the graphics context to set the drawing mode of

***linestyle*** The new style of the line

#### 4.4.2.31 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.

**Parameters:**

***gc*** the ID of the graphics context to set the drawing mode of

***mode*** the new drawing mode

#### 4.4.2.32 void GrSetGCStipple (GR_GC_ID *gc*, GR_BITMAP ∗ *bitmap*, int *width*, int *height*)

FIXME.

**Parameters:**

***gc*** FIXME

***bitmap*** FIXME

***width*** FIXME

***height*** FIXME

**Todo:**

FIXME document this

#### 4.4.2.33 void GrSetGCTile (GR_GC_ID *gc*, GR_WINDOW_ID *pixmap*, int *width*, int *height*)

FIXME.

**Parameters:**

***gc*** FIXME

***pixmap*** FIXME

***width*** FIXME

***height*** FIXME

**Todo:**

FIXME document this

#### 4.4.2.34 void GrSetGCTSOffset (GR_GC_ID *gc*, int *xoff*, int *yoff*)

FIXME.

**Parameters:**

***gc*** FIXME

***xoff*** FIXME

***yoff*** FIXME

**Todo:**

FIXME document this

#### 4.4.2.35 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.

**Parameters:**

***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

#### 4.4.2.36 void GrStretchArea (GR_DRAW_ID *dstid*, GR_GC_ID *gc*, GR_COORD *dx1*, GR_COORD *dy1*, GR_COORD *dx2*, GR_COORD *dy2*, GR_DRAW_ID *srcid*, GR_COORD *sx1*, GR_COORD *sy1*, GR_COORD *sx2*, GR_COORD *sy2*, unsigned long *op*)

Copies a region from one drawable to another.

Can stretch and/or flip the image. The stretch/flip maps (sx1,sy1) in the source to (dx1,dy1) in the destination, and similarly (sx2,sy2) in the source maps to (dx2,dy2) in the destination. Both horizontal and vertical flips are supported.

Note that the bottom and right rows of pixels are excluded - i.e. a target of (0,0)-(2,2) will not draw pixels with y=2 or x=2.

0 is a sensible default ROP code in most cases.

**Parameters:**

***dstid*** the ID of the drawable to copy the area to

***gc*** the ID of the graphics context to use when copying the area

***dx1*** the X coordinate of the first point describing the destination area

***dy1*** the Y coordinate of the first point describing the destination area

***dx2*** the X coordinate of the second point describing the destination area

***dy2*** the Y coordinate of the second point describing the destination area

***srcid*** the ID of the drawable to copy the area from

***sx1*** the X coordinate of the first point describing the source area

***sy1*** the Y coordinate of the first point describing the source area

***sx2*** the X coordinate of the second point describing the source area

***sy2*** the Y coordinate of the second point describing the source area

***op*** the ROP codes to pass to the blitter when performing the copy

## 4.5 Nano-X events API.

The Nano-X event mechanism.

### Functions

- void GrRegisterInput (int fd)

  *Register an extra file descriptor to monitor in the main select() call.*

- void GrUnregisterInput (int fd)

  *Stop monitoring a file descriptor previously registered with GrRegisterInput().*

- void GrPrepareSelect (int ∗maxfd, void ∗rfdset)

  *Prepare for the client to call select().*

- void GrServiceSelect (void ∗rfdset, GR_FNCALLBACKEVENT fncb)

  *Handles events after the client has done a select() call.*

- void GrMainLoop (GR_FNCALLBACKEVENT fncb)

  *An infinite loop that dispatches events.*

- void GrGetNextEvent (GR_EVENT ∗ep)

  *Gets the next event from the event queue.*

- void GrGetNextEventTimeout (GR_EVENT ∗ep, GR_TIMEOUT timeout)

  *Gets the next event from the event queue, with a time limit.*

- int GrPeekEvent (GR_EVENT ∗ep)

  *Gets a copy of the next event on the queue, without actually removing it from the queue.*

- void GrPeekWaitEvent (GR_EVENT ∗ep)

  *Wait until an event is available for a client, and then peek at it.*

- void GrCheckNextEvent (GR_EVENT ∗ep)

  *Gets the next event from the event queue if there is one.*

- int GrGetTypedEvent (GR_WINDOW_ID wid, GR_EVENT_MASK mask, GR_-UPDATE_TYPE update, GR_EVENT ∗ep, GR_BOOL block)

  *Fills in the specified event structure with a copy of the next event on the queue that matches the type parameters passed and removes it from the queue.*

- int GrGetTypedEventPred (GR_WINDOW_ID wid, GR_EVENT_MASK mask, GR_UPDATE_TYPE update, GR_EVENT ∗ep, GR_BOOL block, GR_TYPED_-EVENT_CALLBACK matchfn, void ∗arg)

*The specified callback function is called with the passed event type parameters for each event on the queue, until the callback function CheckFunction returns GR_-TRUE.*

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

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

- int GrQueueLength (void)

  *Returns the current length of the client side queue.*

### 4.5.1 Detailed Description

The Nano-X event mechanism.

### 4.5.2 Function Documentation

#### 4.5.2.1 void GrCheckNextEvent (GR_EVENT ∗ *ep*)

Gets the next event from the event queue if there is one.

Returns immediately with an event type of GR_EVENT_TYPE_NONE if the queue is empty.

**Parameters:**

***ep*** Pointer to the GR_EVENT structure to return the event in.

#### 4.5.2.2 void GrGetNextEvent (GR_EVENT ∗ *ep*)

Gets the next event from the event queue.

If the queue is currently empty, sleeps until the next event arrives from the server or input is read on a file descriptor previously specified by GrRegisterInput().

**Parameters:**

***ep*** Pointer to the GR_EVENT structure to return the event in.

#### 4.5.2.3 void GrGetNextEventTimeout (GR_EVENT ∗ *ep*, GR_TIMEOUT *timeout*)

Gets the next event from the event queue, with a time limit.

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".

**Parameters:**

*ep* Pointer to the GR_EVENT structure to return the event in.

*timeout* The number of milliseconds to wait before timing out, or 0 for forever.

#### 4.5.2.4 int GrGetTypedEvent (GR_WINDOW_ID *wid*, GR_EVENT_MASK *mask*, GR_UPDATE_TYPE *update*, GR_EVENT ∗ *ep*, GR_BOOL *block*)

Fills in the specified event structure with a copy of the next event on the queue that matches the type parameters passed and removes it from the queue.

If block is GR_TRUE, the call will block until a matching event is found. Otherwise, only the local queue is searched, and an event type of GR_EVENT_TYPE_NONE is returned if the a match is not found.

**Parameters:**

*wid* Window id for which to check events. 0 means no window.

*mask* Event mask of events for which to check. 0 means no check for mask.

*update* Update event subtype when event mask is GR_EVENT_MASK_UPDATE.

*ep* Pointer to the GR_EVENT structure to return the event in.

*block* Specifies whether or not to block, GR_TRUE blocks, GR_FALSE does not.

**Returns:**

GR_EVENT_TYPE if an event was returned, or GR_EVENT_TYPE_NONE if no events match.

#### 4.5.2.5 int GrGetTypedEventPred (GR_WINDOW_ID *wid*, GR_EVENT_MASK *mask*, GR_UPDATE_TYPE *update*, GR_EVENT ∗ *ep*, GR_BOOL *block*, GR_TYPED_EVENT_CALLBACK *matchfn*, void ∗ *arg*)

The specified callback function is called with the passed event type parameters for each event on the queue, until the callback function CheckFunction returns GR_TRUE.

The event is then removed from the queue and returned. If block is GR_TRUE, the call will block until a matching event is found. Otherwise, only the local queue is searched, and an event type of GR_EVENT_TYPE_NONE is returned if the a match is not found.

**Parameters:**

*wid* Window id for which to check events. 0 means no window.

*mask* Event mask of events for which to check. 0 means no check for mask.

*update* Update event subtype when event mask is GR_EVENT_MASK_UPDATE.

*ep* Pointer to the GR_EVENT structure to return the event in.

*block* Specifies whether or not to block, GR_TRUE blocks, GR_FALSE does not.

***matchfn*** Specifies the callback function called for matching.

***arg*** A programmer-specified argument passed to the callback function.

**Returns:**

GR_EVENT_TYPE if an event was returned, or GR_EVENT_TYPE_NONE if no events match.

#### 4.5.2.6 void GrMainLoop (GR_FNCALLBACKEVENT *fncb*)

An infinite loop that dispatches events.

Calls the specified callback function whenever an event arrives or there is data to be read on a file descriptor registered with GrRegisterInput(). Never returns.

**Parameters:**

***fncb*** Pointer to the function to call when an event needs handling.

#### 4.5.2.7 int GrPeekEvent (GR_EVENT ∗ *ep*)

Gets a copy of the next event on the queue, without actually removing it from the queue.

Does not block - an event type of GR_EVENT_TYPE_NONE is given if the queue is empty.

**Parameters:**

***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.

#### 4.5.2.8 void GrPeekWaitEvent (GR_EVENT ∗ *ep*)

Wait until an event is available for a client, and then peek at it.

**Parameters:**

***ep*** Pointer to the GR_EVENT structure to return the event in.

#### 4.5.2.9 void GrPrepareSelect (int ∗ *maxfd*, void ∗ *rfdset*)

Prepare for the client to call select().

Asks the server to send the next event but does not wait around for it to arrive. Initializes 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.

Usually used in conjunction with GrServiceSelect().

Note that in a multithreaded client, the application must ensure that no Nano-X calls are made between the calls to GrPrepareSelect() and GrServiceSelect(), else there will be race conditions.

**Parameters:**

***maxfd*** Pointer to a variable which the highest in use fd will be written to. Must contain a valid value on input - will only be overwritten if the new value is higher than the old value.

***rfdset*** Pointer to the file descriptor set structure to use. Must be valid on input - file descriptors will be added to this set without clearing the previous contents.

#### 4.5.2.10 int GrQueueLength (void)

Returns the current length of the client side queue.

**Returns:**

The current length of the client side queue.

#### 4.5.2.11 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.

**Parameters:**

***fd*** The file descriptor to monitor.

#### 4.5.2.12 void GrSelectEvents (GR_WINDOW_ID *wid*, GR_EVENT_MASK *eventmask*)

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

**Parameters:**

***wid*** The ID of the window to set the event mask of.

***eventmask*** A bit field specifying the desired event mask.

#### 4.5.2.13 void GrServiceSelect (void ∗ *rfdset*, GR_FNCALLBACKEVENT *fncb*)

Handles events after the client has done a select() call.

Calls the specified callback function is an event has arrived, or if there is data waiting on an external fd specified by GrRegisterInput().

Used by GrMainLoop().

**Parameters:**

***rfdset*** Pointer to the file descriptor set containing those file descriptors that are ready for reading.

***fncb*** Pointer to the function to call when an event needs handling.

#### 4.5.2.14 void GrUnregisterInput (int *fd*)

Stop monitoring a file descriptor previously registered with GrRegisterInput().

**Parameters:**

***fd*** The file descriptor to stop monitoring.

## 4.6 Nano-X font API.

Functions for handling fonts and drawing text.

### Functions

- void GrGetFontInfo (GR_FONT_ID font, GR_FONT_INFO ∗fip)

  *Gets information about a font.*

- void GrGetGCTextSize (GR_GC_ID gc, void ∗str, int count, GR_TEXTFLAGS flags, GR_SIZE ∗retwidth, GR_SIZE ∗retheight, GR_SIZE ∗retbase)

  *Calculates the dimensions of a specified text string.*

- GR_FONT_ID GrCreateFont (GR_CHAR ∗name, GR_COORD height, GR_LOGFONT ∗plogfont)

  *Attempts to locate a font with the desired attributes and returns a font ID number which can be used to refer to it.*

- void GrGetFontList (GR_FONTLIST ∗∗∗fonts, int ∗numfonts)

  *Returns an array of strings containing the names of available fonts and an integer that specifies the number of strings returned.*

- void GrFreeFontList (GR_FONTLIST ∗∗∗fonts, int numfonts)

  *Frees the specified font list array.*

- void GrSetFontSize (GR_FONT_ID fontid, GR_COORD size)

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

- void GrSetFontRotation (GR_FONT_ID fontid, int tenthsdegrees)

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

- 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.*

- 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.*

- 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.*

- void GrText (GR_DRAW_ID id, GR_GC_ID gc, GR_COORD x, GR_COORD y, void ∗str, GR_COUNT count, GR_TEXTFLAGS flags)

  *Draws the specified text string at the specified position on the specified drawable using the specified graphics context and flags.*

### 4.6.1 Detailed Description

Functions for handling fonts and drawing text.

### 4.6.2 Function Documentation

#### 4.6.2.1 GR_FONT_ID GrCreateFont (GR_CHAR ∗ *name*, GR_COORD *height*, GR_LOGFONT ∗ *plogfont*)

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.

**Parameters:**

***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

#### 4.6.2.2 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.

**Parameters:**

***fontid*** the ID of the font to destroy

#### 4.6.2.3 void GrFreeFontList (GR_FONTLIST ∗∗∗ *fonts*, int *numfonts*)

Frees the specified font list array.

**Parameters:**

***fonts*** Pointer to array returned by GrGetFontList().

***numfonts*** The number of font names in the array.

#### 4.6.2.4 void GrGetFontInfo (GR_FONT_ID *font*, GR_FONT_INFO ∗ *fip*)

Gets information about a font.

**Parameters:**

***font*** The font ID to query.

***fip*** Pointer to the GR_FONT_INFO structure to store the result.

#### 4.6.2.5 void GrGetFontList (GR_FONTLIST ∗∗∗ *fonts*, int ∗ *numfonts*)

Returns an array of strings containing the names of available fonts and an integer that specifies the number of strings returned.

**Parameters:**

***fonts*** pointer used to return an array of font names.

***numfonts*** pointer used to return the number of names found.

#### 4.6.2.6 void GrGetGCTextSize (GR_GC_ID *gc*, void ∗ *str*, int *count*, GR_TEXTFLAGS *flags*, GR_SIZE ∗ *retwidth*, GR_SIZE ∗ *retheight*, GR_SIZE ∗ *retbase*)

Calculates the dimensions of a specified text string.

Uses the current font and flags in the specified graphics context. The count argument can be -1 if the string is null terminated.

**Parameters:**

***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.

#### 4.6.2.7 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.

**Parameters:**

***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

#### 4.6.2.8 void GrSetFontRotation (GR_FONT_ID *fontid*, int *tenthdegrees*)

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

**Parameters:**

***fontid*** the ID number of the font to rotate

***tenthdegrees*** the angle to set the rotation to in tenths of a degree

#### 4.6.2.9 void GrSetFontSize (GR_FONT_ID *fontid*, GR_COORD *size*)

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

**Parameters:**

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

***size*** the size to change the font to

#### 4.6.2.10 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.

**Parameters:**

***gc*** the ID of the graphics context to set the font of

***font*** the ID of the font

#### 4.6.2.11 void GrText (GR_DRAW_ID *id*, GR_GC_ID *gc*, GR_COORD *x*, GR_COORD *y*, void ∗ *str*, GR_COUNT *count*, GR_TEXTFLAGS *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.

**Parameters:**

***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.

## 4.7 Nano-X basic API.

Functions to initialise and close Nano-X.

### Functions

- int GrOpen (void)

  *Open a connection to the graphics server.*

- void GrClose (void)

  *Close the graphics device.*

- void GrFlush (void)

  *Flush the message buffer of any messages it may contain.*

- void GrDefaultErrorHandler (GR_EVENT ∗ep)

  *The default error handler.*

- GR_FNCALLBACKEVENT GrSetErrorHandler (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).*

- void GrGetScreenInfo (GR_SCREEN_INFO ∗sip)

  *Fills in the specified GR_SCREEN_INFO structure.*

### 4.7.1 Detailed Description

Functions to initialise and close Nano-X.

### 4.7.2 Function Documentation

#### 4.7.2.1 void GrClose (void)

Close the graphics device.

Flushes any waiting messages.

#### 4.7.2.2 void GrDefaultErrorHandler (GR_EVENT ∗ *ep*)

The default error handler.

This 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 describing what error occurred and what function it occured in, then exits.

**Parameters:**
*ep* The error event structure.

#### 4.7.2.3 void GrGetScreenInfo (GR_SCREEN_INFO ∗ *sip*)

Fills in the specified GR_SCREEN_INFO structure.

**Parameters:**
*sip* Pointer to a GR_SCREEN_INFO structure

#### 4.7.2.4 int GrOpen (void)

Open a connection to the graphics server.

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

#### 4.7.2.5 GR_FNCALLBACKEVENT GrSetErrorHandler (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.

**Parameters:**
*fncb* the function to call to handle error events

**Returns:**
the address of the previous error handler

## 4.8 Nano-X image file API.

Functions to draw images from standard image file formats.

### Functions

- 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)

  *Draws the image from the specified image buffer at the specified position on the specified drawable using the specified graphics context.*

- void GrFreeImage (GR_IMAGE_ID id)

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

- 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.*

- GR_IMAGE_ID GrLoadImageFromBuffer (void ∗buffer, int size, int flags)

  *FIXME.*

- void GrDrawImageFromBuffer (GR_DRAW_ID id, GR_GC_ID gc, GR_COORD x, GR_COORD y, GR_SIZE width, GR_SIZE height, void ∗buffer, int size, int flags)

  *FIXME.*

### 4.8.1 Detailed Description

Functions to draw images from standard image file formats.

### 4.8.2 Function Documentation

#### 4.8.2.1 void GrDrawImageFromBuffer (GR_DRAW_ID *id*, GR_GC_ID *gc*, GR_COORD *x*, GR_COORD *y*, GR_SIZE *width*, GR_SIZE *height*, void ∗ *buffer*, int *size*, int *flags*)

FIXME.

**Parameters:**

***id*** FIXME

***gc*** FIXME

***x*** FIXME

***y*** FIXME

***width*** FIXME

***height*** FIXME

***buffer*** FIXME

***size*** FIXME

***flags*** FIXME

**Todo:**

FIXME document this

#### 4.8.2.2 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*)

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.

**Parameters:**

***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

#### 4.8.2.3 void GrFreeImage (GR_IMAGE_ID *id*)

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

**Parameters:**

***id*** ID of the image buffer to free

#### 4.8.2.4 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.

**Parameters:**

***id*** ID of an image buffer

***iip*** pointer to a GR_IMAGE_INFO structure

#### 4.8.2.5 GR_IMAGE_ID GrLoadImageFromBuffer (void * *buffer*, int *size*, int *flags*)

FIXME.

**Parameters:**

***buffer*** FIXME

***size*** FIXME

***flags*** FIXME

**Todo:**

FIXME document this

## 4.9 Nano-X miscellaneous APIs.

Functions that didn't fit anywhere else.

### Functions

- void GrReqShmCmds (long shmsize)

  *Requests a shared memory area of the specified size to use for transferring command arguments.*

- void GrInjectPointerEvent (GR_COORD x, GR_COORD 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.*

- void GrInjectKeyboardEvent (GR_WINDOW_ID wid, GR_KEY keyvalue, GR_-KEYMOD modifiers, GR_SCANCODE scancode, GR_BOOL pressed)

  *Sends a keyboard event to the specified window, or to the window with the current keyboard focus if 0 is used as the ID.*

- void GrSetScreenSaverTimeout (GR_TIMEOUT timeout)

  *Sets the number of seconds of inactivity before a screen saver activate event is sent to the root window ID.*

- 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).*

- void GrSetPortraitMode (int portraitmode)

  *Set server portrait mode.*

- void GrQueryPointer (GR_WINDOW_ID ∗mwin, GR_COORD ∗x, GR_-COORD ∗y, GR_BUTTON ∗bmask)

  *Returns the current information for the pointer.*

- GR_BOOL GrGrabKey (GR_WINDOW_ID wid, GR_KEY key, int type)

  *Grab a key for a specific window.*

- void GrUngrabKey (GR_WINDOW_ID wid, GR_KEY key)

  *Ungrab a key for a specific window.*

- void GrSetTransform (GR_TRANSFORM ∗)

  *This passes transform data to the mouse input engine.*

### 4.9.1 Detailed Description

Functions that didn't fit anywhere else.

### 4.9.2 Function Documentation

#### 4.9.2.1 GR_BOOL GrGrabKey (GR_WINDOW_ID *id*, GR_KEY *key*, int *type*)

Grab a key for a specific window.

This function has two effects. With any type other than GR_GRAB_HOTKEY it attempts to reserve the specified key for exclusive use by the application. In addition, with GR_GRAB_HOTKEY or GR_GRAB_HOTKEY_EXCLUSIVE it requests hotkey events be sent to the specified window whenever the specified key is pressed or released.

A key can have any number of reservations of type GR_GRAB_HOTKEY, but at most one reservation of another type. This means that grabs of type GR_GRAB_HOTKEY always succeed, but grabs of any other type will fail if the key is already grabbed in any fashion except GR_GRAB_HOTKEY.

Note that all grabs are automatically released when the window specified in the id paramater is deleted, or when the client application closes it's connection to the Nano-X server.

**Parameters:**

***id*** Window to send event to.

***key*** MWKEY value.

***type*** The type of reservation to make. Valid values are GR_GRAB_HOTKEY_EXCLUSIVE, GR_GRAB_HOTKEY, GR_GRAB_EXCLUSIVE and GR_GRAB_EXCLUSIVE_MOUSE.

**Returns:**

GR_TRUE on success, GR_FALSE on error.

#### 4.9.2.2 void GrInjectKeyboardEvent (GR_WINDOW_ID *wid*, GR_KEY *keyvalue*, GR_KEYMOD *modifiers*, GR_SCANCODE *scancode*, GR_BOOL *pressed*)

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.

**Parameters:**

***wid*** ID of the window to send the event to, or 0.

***keyvalue*** Unicode keystroke value to inject.

***modifiers*** Modifiers (shift, ctrl, alt, etc.) to inject.

***scancode*** The key scan code to inject.

***pressed*** TRUE for a key press, FALSE for a key release.

#### 4.9.2.3 void GrInjectPointerEvent (GR_COORD *x*, GR_COORD *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.

**Parameters:**

***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

#### 4.9.2.4 void GrQueryPointer (GR_WINDOW_ID ∗ *mwin*, GR_COORD ∗ *x*, GR_COORD ∗ *y*, GR_BUTTON ∗ *bmask*)

Returns the current information for the pointer.

**Parameters:**

***mwin*** Window the mouse is current in

***x*** Current X pos of mouse (from root)

***y*** Current Y pos of mouse (from root)

***bmask*** Current button mask

#### 4.9.2.5 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.

**Parameters:**

***shmsize*** the size of the shared memory area to allocate

**Todo:**

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

#### 4.9.2.6 void GrSetPortraitMode (int *portraitmode*)

Set server portrait mode.

**Parameters:**

***portraitmode*** New portrait mode.

#### 4.9.2.7 void GrSetScreenSaverTimeout (GR_TIMEOUT *timeout*)

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.

**Parameters:**

***timeout*** the number of seconds of inactivity before screen saver activates

#### 4.9.2.8 void GrSetTransform (GR_TRANSFORM ∗ *trans*)

This passes transform data to the mouse input engine.

**Parameters:**

***trans*** A GR_TRANSFORM structure that contains the transform data for the filter, or NULL to disable.

#### 4.9.2.9 void GrUngrabKey (GR_WINDOW_ID *id*, GR_KEY *key*)

Ungrab a key for a specific window.

**Parameters:**

***id*** Window to stop key grab.

***key*** MWKEY value.

## 4.10 Nano-X region API.

Functions for handling clipping regions - these are used for clipping drawing, and for non-rectangular windows.

### Functions

- GR_REGION_ID GrNewRegion (void)

  *Creates a new region structure.*

- void GrDestroyRegion (GR_REGION_ID region)

  *Destroys a region structure.*

- void GrUnionRectWithRegion (GR_REGION_ID region, GR_RECT ∗rect)

  *Makes a union of the specified region and the specified rectangle.*

- void GrUnionRegion (GR_REGION_ID dst_rgn, GR_REGION_ID src_rgn1, GR_REGION_ID src_rgn2)

  *Makes a union of two regions.*

- void GrSubtractRegion (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.*

- void GrXorRegion (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.*

- void GrIntersectRegion (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.*

- void GrSetGCRegion (GR_GC_ID gc, GR_REGION_ID region)

  *Sets the clip mask of the specified graphics context to the specified region.*

- GR_BOOL GrPointInRegion (GR_REGION_ID region, 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.*

- 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.*

- GR_BOOL GrEmptyRegion (GR_REGION_ID region)

  *Determines whether the specified region is empty.*

- 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.*

- void GrOffsetRegion (GR_REGION_ID region, GR_SIZE dx, GR_SIZE dy)

  *Offsets the specified region by the specified distance.*

- int GrGetRegionBox (GR_REGION_ID region, 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.*

- 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.*

- GR_REGION_ID GrNewBitmapRegion (GR_BITMAP ∗bitmap, GR_SIZE width, GR_SIZE height)

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

## 4.10.1 Detailed Description

Functions for handling clipping regions - these are used for clipping drawing, and for non-rectangular windows.

## 4.10.2 Function Documentation

### 4.10.2.1 void GrDestroyRegion (GR_REGION_ID *region*)

Destroys a region structure.

**Parameters:**

***region*** The ID of the region structure to destroy.

### 4.10.2.2 GR_BOOL GrEmptyRegion (GR_REGION_ID *region*)

Determines whether the specified region is empty.

**Parameters:**

***region*** The ID of the region to examine.

**Returns:**

GR_TRUE if the region is empty, or GR_FALSE if it is not.

#### 4.10.2.3 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.

**Parameters:**

***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

#### 4.10.2.4 int GrGetRegionBox (GR_REGION_ID *region*, 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.

**Parameters:**

***region*** The ID of the region to get the bounding box of

***rect*** Pointer to a rectangle structure

**Returns:**

The region type

**Todo:**

FIXME check Doxygen comments from this point down.

#### 4.10.2.5 void GrIntersectRegion (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.

**Parameters:**

***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.

#### 4.10.2.6 GR_REGION_ID GrNewBitmapRegion (GR_BITMAP ∗ *bitmap*, GR_SIZE *width*, GR_SIZE *height*)

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

1 bits in the bitmap specify areas inside the region and 0 bits specify areas outside it.

**Parameters:**

***bitmap*** pointer to a GR_BITMAP array specifying the region mask

***width*** the width of the bitmap

***height*** the height of the bitmap

**Returns:**

the ID of the newly allocated region structure, or 0 on error

#### 4.10.2.7 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.

**Parameters:**

***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

#### 4.10.2.8 GR_REGION_ID GrNewRegion (void)

Creates a new region structure.

The structure is initialised with a set of default parameters.

**Returns:**

the ID of the newly created region

#### 4.10.2.9 void GrOffsetRegion (GR_REGION_ID *region*, GR_SIZE *dx*, GR_SIZE *dy*)

Offsets the specified region by the specified distance.

**Parameters:**

***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

#### 4.10.2.10 GR_BOOL GrPointInRegion (GR_REGION_ID *region*, 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.

**Parameters:**

***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, otherwise FALSE.

#### 4.10.2.11 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.

**Parameters:**

***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.

#### 4.10.2.12 void GrSetGCRegion (GR_GC_ID *gc*, GR_REGION_ID *region*)

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.

**Parameters:**

***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, or 0 for none.

#### 4.10.2.13 void GrSubtractRegion (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.

**Parameters:**

***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.

#### 4.10.2.14 void GrUnionRectWithRegion (GR_REGION_ID *region*, GR_RECT ∗ *rect*)

Makes a union of the specified region and the specified rectangle.

Places the result back in the source region.

**Parameters:**

***region*** The ID of the region to modify.

***rect*** A pointer to the rectangle to add to the region.

#### 4.10.2.15 void GrUnionRegion (GR_REGION_ID *dst_rgn*, GR_REGION_ID *src_rgn1*, GR_REGION_ID *src_rgn2*)

Makes a union of two regions.

Places the result in the specified destination region.

**Parameters:**

***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.

#### 4.10.2.16 void GrXorRegion (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.

**Parameters:**

- ***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.

## 4.11 Nano-X clipboard API.

Functions for handling the current selection on the clipboard.

### Functions

- void GrSetSelectionOwner (GR_WINDOW_ID wid, GR_CHAR ∗typelist)

  *Sets the current selection (otherwise known as the clipboard) ownership to the specified window.*

- GR_WINDOW_ID GrGetSelectionOwner (GR_CHAR ∗∗typelist)

  *Finds the window which currently owns the selection and returns its ID, or 0 if no window currently owns the selection.*

- void GrRequestClientData (GR_WINDOW_ID wid, GR_WINDOW_ID rid, GR_SERIALNO serial, GR_MIMETYPE mimetype)

  *Sends a CLIENT_DATA_REQ event to the specified window.*

- void GrSendClientData (GR_WINDOW_ID wid, GR_WINDOW_ID did, GR_SERIALNO serial, GR_LENGTH len, GR_LENGTH thislen, void ∗data)

  *Used as the response to a CLIENT_DATA_REQ event.*

### 4.11.1 Detailed Description

Functions for handling the current selection on the clipboard.

### 4.11.2 Function Documentation

#### 4.11.2.1 GR_WINDOW_ID GrGetSelectionOwner (GR_CHAR ∗∗ *typelist*)

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.

**Parameters:**

***typelist*** pointer used to return the list of available mime types

**Returns:**

the ID of the window which currently owns the selection, or 0

#### 4.11.2.2 void GrRequestClientData (GR_WINDOW_ID *wid*, GR_WINDOW_ID *rid*, GR_SERIALNO *serial*, GR_MIMETYPE *mimetype*)

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.

**Parameters:**

***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

#### 4.11.2.3 void GrSendClientData (GR_WINDOW_ID *wid*, GR_WINDOW_ID *did*, GR_SERIALNO *serial*, GR_LENGTH *len*, GR_LENGTH *thislen*, 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. The thislen parameter is used internally to split the data up into packets. It should be set to the same value as the len parameter.

**Parameters:**

***wid*** The ID of the window sending the data.

***did*** The ID of the destination window.

***serial*** The serial number of the request.

***len*** Number of bytes of data to transfer.

***thislen*** Number of bytes in this packet.

***data*** Pointer to the data to transfer.

#### 4.11.2.4 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).

**Parameters:**

***wid*** the ID of the window to set the selection owner to

***typelist*** list of mime types selection data can be supplied as

## 4.12 Nano-X timer API.

Functions for handling timers and delays.

### Functions

- void GrDelay (GR_TIMEOUT msecs)

  *This function suspends execution of the program for the specified number of milliseconds.*

### 4.12.1 Detailed Description

Functions for handling timers and delays.

### 4.12.2 Function Documentation

#### 4.12.2.1 void GrDelay (GR_TIMEOUT *msecs*)

This function suspends execution of the program for the specified number of milliseconds.

**Parameters:**

***msecs*** Number of milliseconds to delay.

## 4.13 Nano-X window API.

Functions for handling windows on the screen.

### Functions

- 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.*

- GR_WINDOW_ID GrNewPixmap (GR_SIZE width, GR_SIZE height, void ∗pixels)

  *Create a new server side pixmap.*

- GR_WINDOW_ID GrNewInputWindow (GR_WINDOW_ID parent, 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.*

- void GrDestroyWindow (GR_WINDOW_ID wid)

  *Destroys a window and all of it's children.*

- void GrGetWindowInfo (GR_WINDOW_ID wid, GR_WINDOW_INFO ∗infoptr)

  *Fills in a GR_WINDOW_INFO structure with information regarding a window.*

- 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.*

- void GrUnmapWindow (GR_WINDOW_ID wid)

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

- void GrRaiseWindow (GR_WINDOW_ID wid)

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

- void GrLowerWindow (GR_WINDOW_ID wid)

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

- 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.*

- void GrResizeWindow (GR_WINDOW_ID wid, GR_SIZE width, GR_SIZE height)

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

- 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.*

- GR_WINDOW_ID GrGetFocus (void)

  *Returns the ID of the window which currently has the keyboard focus.*

- void GrSetFocus (GR_WINDOW_ID wid)

  *Sets the keyboard focus to the specified window.*

- 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.*

- 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.*

- 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.*

- void GrKillWindow (GR_WINDOW_ID wid)

  *Forcibly disconnects the client which owns this window with the specified ID number.*

- void GrSetBackgroundPixmap (GR_WINDOW_ID wid, GR_WINDOW_ID pixmap, int flags)

  *Sets the background of the specified window to the specified pixmap.*

- void GrQueryTree (GR_WINDOW_ID wid, GR_WINDOW_ID ∗parentid, GR_-WINDOW_ID ∗∗children, GR_COUNT ∗nchildren)

  *Return window parent and list of children.*

- void GrSetWindowRegion (GR_WINDOW_ID wid, GR_REGION_ID rid, int type)

  *Sets the bounding region of the specified window, not to be confused with a GC clip region.*

### 4.13.1 Detailed Description

Functions for handling windows on the screen.

### 4.13.2 Function Documentation

#### 4.13.2.1 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.

**Parameters:**

***wid*** the ID of the window to send the CLOSE_REQ event to

#### 4.13.2.2 void GrDestroyWindow (GR_WINDOW_ID *wid*)

Destroys a window and all of it's children.

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

**Parameters:**

***wid*** The ID of the window to destroy.

#### 4.13.2.3 GR_WINDOW_ID GrGetFocus (void)

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

**Returns:**

the ID of the window which currently has the keyboard focus

#### 4.13.2.4 void GrGetWindowInfo (GR_WINDOW_ID *wid*, GR_WINDOW_INFO ∗ *infoptr*)

Fills in a GR_WINDOW_INFO structure with information regarding a window.

**Parameters:**

***wid*** The ID of the window to retrieve information about.

***infoptr*** Pointer to a GR_WINDOW_INFO structure to return the information in.

#### 4.13.2.5 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.

**Parameters:**

***wid*** the ID of the window to retreive the WM properties of

***props*** pointer to a GR_WM_PROPERTIES structure to fill in

#### 4.13.2.6 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.

**Parameters:**

***wid*** the ID of the window to kill

#### 4.13.2.7 void GrLowerWindow (GR_WINDOW_ID *wid*)

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

**Parameters:**

***wid*** the ID of the window to lower

#### 4.13.2.8 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.

**Parameters:**

***wid*** the ID of the window to map

#### 4.13.2.9 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.

**Parameters:**

***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.

#### 4.13.2.10 GR_WINDOW_ID GrNewInputWindow (GR_WINDOW_ID *parent*, 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.

**Parameters:**

***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.

#### 4.13.2.11 GR_WINDOW_ID GrNewPixmap (GR_SIZE *width*, GR_SIZE *height*, void ∗ *pixels*)

Create a new server side pixmap.

This is an offscreen drawing area which can be copied into a window using a GrCopy-Area call.

**Parameters:**

***width*** The width of the pixmap.

***height*** The height of the pixmap.

***pixels*** Currently unused in client/server mode.

**Returns:**

The ID of the newly created pixmap.

**Todo:**

FIXME Add support for shared memory...

#### 4.13.2.12 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.

**Parameters:**

***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 color of the window background.

***bordercolor*** The color of the window border.

**Returns:**

The ID of the newly created window.

#### 4.13.2.13 void GrQueryTree (GR_WINDOW_ID *wid*, GR_WINDOW_ID * *parentid*, GR_WINDOW_ID ** *children*, GR_COUNT * *nchildren*)

Return window parent and list of children.

Caller must free() children list after use.

**Parameters:**

***wid*** window ID for query

***parentid*** returned parent ID

***children*** returned children ID list

***nchildren*** returned children count

#### 4.13.2.14 void GrRaiseWindow (GR_WINDOW_ID *wid*)

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

**Parameters:**

***wid*** the ID of the window to raise

#### 4.13.2.15 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.

**Parameters:**

- ***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

#### 4.13.2.16 void GrResizeWindow (GR_WINDOW_ID *wid*, GR_SIZE *width*, GR_SIZE *height*)

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

**Parameters:**

- ***wid*** the ID of the window to resize
- ***width*** the width to resize the window to
- ***height*** the height to resize the window to

#### 4.13.2.17 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.

**Parameters:**

- ***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

#### 4.13.2.18 void GrSetFocus (GR_WINDOW_ID *wid*)

Sets the keyboard focus to the specified window.

**Parameters:**

- ***wid*** the ID of the window to set the focus to

#### 4.13.2.19 void GrSetWindowRegion (GR_WINDOW_ID *wid*, GR_REGION_ID *rid*, int *type*)

Sets the bounding region of the specified window, not to be confused with a GC clip region.

The bounding region is used to implement non-rectangular windows. A window is defined by two regions: the bounding region and the clip region. The bounding region defines the area within the parent window that the window will occupy, including border. The clip region is the subset of the bounding region that is available for subwindows and graphics. The area between the bounding region and the clip region is defined to be the border of the window. Currently, only the window bounding region is implemented.

Copies the specified region and makes the copy be the bounding region used for the specified window. After setting the clipping region, all drawing within the window will be clipped to the specified region (including the drawing of the window background by the server), and mouse events will pass through parts of the window which are outside the clipping region to whatever is underneath them. Also, windows underneath the areas which are outside the clipping region will be able to draw to the screen as if those areas of the window were not there (in other words, you can see through the gaps in the window). This is most commonly used to implement shaped windows (ie. windows which are some shape other than a simple rectangle). Note that if you are using this feature you will probably want to disable window manager decorations so that the window manager does not draw its own container window behind yours and spoil the desired effect. Also note that shaped windows must always have a border size of 0. If you need a border around a shaped window, add it to the clipping region and draw it yourself.

**Parameters:**

***wid*** the ID of the window to set the clipping region of

***rid*** the ID of the region to assign to the specified window

***type*** region type, bounding or clip mask

#### 4.13.2.20 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.

**Parameters:**

***wid*** the ID of the window to set the WM properties of

***props*** pointer to a GR_WM_PROPERTIES structure

#### 4.13.2.21 void GrUnmapWindow (GR_WINDOW_ID *wid*)

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

**Parameters:**

***wid*** the ID of the window to unmap

# Chapter 5

# Microwindows Nano-X API Data Structure Documentation

## 5.1 GR_CAL_DATA Struct Reference

Calibration data passed to GrCalcTransform.

### Data Fields

- int xres

  *X resolution of the screen.*

- int yres

  *Y resolution of the screen.*

- int minx

  *min raw X value*

- int miny

  *min raw Y values*

- int maxx

  *max raw X value*

- int maxy

  *max raw Y value*

- GR_BOOL xswap

  *true if the x component should be swapped*

- GR_BOOL yswap

*true if the y component should be swapped*

### 5.1.1 Detailed Description

Calibration data passed to GrCalcTransform.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.2 GR_EVENT Union Reference

Union of all possible event structures.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_EVENT_ERROR error

  *error event*

- GR_EVENT_GENERAL general

  *general window events*

- GR_EVENT_BUTTON button

  *button events*

- GR_EVENT_KEYSTROKE keystroke

  *keystroke events*

- GR_EVENT_EXPOSURE exposure

  *exposure events*

- GR_EVENT_MOUSE mouse

  *mouse motion events*

- GR_EVENT_FDINPUT fdinput

  *fd input events*

- GR_EVENT_UPDATE update

  *window update events*

- GR_EVENT_SCREENSAVER screensaver

  *Screen saver events.*

- GR_EVENT_CLIENT_DATA_REQ clientdatareq

  *Request for client data events.*

- GR_EVENT_CLIENT_DATA clientdata

  *Client data events.*

- GR_EVENT_SELECTION_CHANGED selectionchanged

  *Selection owner changed.*

- GR_EVENT_TIMER timer

*Timer events.*

### 5.2.1 Detailed Description

Union of all possible event structures.

This is the structure returned by GrGetNextEvent() and similar routines.

The documentation for this union was generated from the following file:

- nano-X.h

## 5.3 GR_EVENT_BUTTON Struct Reference

Event for a mouse button pressed down or released.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_WINDOW_ID wid

  *window id event delivered to*

- GR_WINDOW_ID subwid

  *sub-window id (pointer was in)*

- GR_COORD rootx

  *root window x coordinate*

- GR_COORD rooty

  *root window y coordinate*

- GR_COORD x

  *window x coordinate of mouse*

- GR_COORD y

  *window y coordinate of mouse*

- GR_BUTTON buttons

  *current state of all buttons*

- GR_BUTTON changebuttons

  *buttons which went down or up*

- GR_KEYMOD modifiers

  *modifiers (MWKMOD_SHIFT, etc)*

- GR_TIMEOUT time

  *tickcount time value*

### 5.3.1 Detailed Description

Event for a mouse button pressed down or released.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.4 GR_EVENT_CLIENT_DATA Struct Reference

GR_EVENT_TYPE_CLIENT_DATA.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_WINDOW_ID wid

  *ID of window data is destined for.*

- GR_WINDOW_ID rid

  *ID of window data is from.*

- GR_SERIALNO serial

  *Serial number of transaction.*

- unsigned long len

  *Total length of data.*

- unsigned long datalen

  *Length of following data.*

- void $*$ data

  *Pointer to data (filled in on client side).*

### 5.4.1 Detailed Description

GR_EVENT_TYPE_CLIENT_DATA.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.5 GR_EVENT_CLIENT_DATA_REQ Struct Reference

GR_EVENT_TYPE_CLIENT_DATA_REQ.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_WINDOW_ID wid

  *ID of requested window.*

- GR_WINDOW_ID rid

  *ID of window to send data to.*

- GR_SERIALNO serial

  *Serial number of transaction.*

- GR_MIMETYPE mimetype

  *Type to supply data as.*

### 5.5.1 Detailed Description

GR_EVENT_TYPE_CLIENT_DATA_REQ.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.6 GR_EVENT_ERROR Struct Reference

Event for errors detected by the server.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_FUNC_NAME name

  *function name which failed*

- GR_ERROR code

  *error code*

- GR_ID id

  *resource id (maybe useless)*

### 5.6.1 Detailed Description

Event for errors detected by the server.

These events are not delivered to GrGetNextEvent, but instead call the user supplied error handling function. Only the first one of these errors at a time is saved for delivery to the client since there is not much to be done about errors anyway except complain and exit.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.7 GR_EVENT_EXPOSURE Struct Reference

Event for exposure for a region of a window.

### Data Fields

- GR_EVENT_TYPE type

  *event type*
- GR_WINDOW_ID wid

  *window id*
- GR_COORD x

  *window x coordinate of exposure*
- GR_COORD y

  *window y coordinate of exposure*
- GR_SIZE width

  *width of exposure*
- GR_SIZE height

  *height of exposure*

### 5.7.1 Detailed Description

Event for exposure for a region of a window.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.8 GR_EVENT_FDINPUT Struct Reference

GrRegisterInput() event.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- int fd

  *input file descriptor*

### 5.8.1 Detailed Description

GrRegisterInput() event.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.9 GR_EVENT_GENERAL Struct Reference

General events for focus in or focus out for a window, or mouse enter or mouse exit from a window, or window unmapping or mapping, etc.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_WINDOW_ID wid

  *window id*

- GR_WINDOW_ID otherid

  *new/old focus id for focus events*

### 5.9.1 Detailed Description

General events for focus in or focus out for a window, or mouse enter or mouse exit from a window, or window unmapping or mapping, etc.

Server portrait mode changes are also sent using this event to all windows that request it.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.10 GR_EVENT_KEYSTROKE Struct Reference

Event for a keystroke typed for the window with has focus.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_WINDOW_ID wid

  *window id event delived to*

- GR_WINDOW_ID subwid

  *sub-window id (pointer was in)*

- GR_COORD rootx

  *root window x coordinate*

- GR_COORD rooty

  *root window y coordinate*

- GR_COORD x

  *window x coordinate of mouse*

- GR_COORD y

  *window y coordinate of mouse*

- GR_BUTTON buttons

  *current state of buttons*

- GR_KEYMOD modifiers

  *modifiers (MWKMOD_SHIFT, etc)*

- GR_KEY ch

  *16-bit unicode key value, MWKEY_xxx*

- GR_SCANCODE scancode

  *OEM scancode value if available.*

- GR_BOOL hotkey

  *TRUE if generated from GrGrabKey(GR_GRAB_HOTKEY_x).*

### 5.10.1 Detailed Description

Event for a keystroke typed for the window with has focus.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.11 GR_EVENT_MOUSE Struct Reference

Events for mouse motion or mouse position.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_WINDOW_ID wid

  *window id event delivered to*

- GR_WINDOW_ID subwid

  *sub-window id (pointer was in)*

- GR_COORD rootx

  *root window x coordinate*

- GR_COORD rooty

  *root window y coordinate*

- GR_COORD x

  *window x coordinate of mouse*

- GR_COORD y

  *window y coordinate of mouse*

- GR_BUTTON buttons

  *current state of buttons*

- GR_KEYMOD modifiers

  *modifiers (MWKMOD_SHIFT, etc)*

### 5.11.1 Detailed Description

Events for mouse motion or mouse position.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.12 GR_EVENT_SCREENSAVER Struct Reference

GR_EVENT_TYPE_SCREENSAVER.

### Data Fields

- GR_EVENT_TYPE type

    *event type*

- GR_BOOL activate

    *true = activate, false = deactivate*

### 5.12.1 Detailed Description

GR_EVENT_TYPE_SCREENSAVER.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.13 GR_EVENT_SELECTION_CHANGED Struct Reference

GR_EVENT_TYPE_SELECTION_CHANGED.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_WINDOW_ID new_owner

  *ID of new selection owner.*

### 5.13.1 Detailed Description

GR_EVENT_TYPE_SELECTION_CHANGED.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.14 GR_EVENT_TIMER Struct Reference

GR_EVENT_TYPE_TIMER.

### Data Fields

- GR_EVENT_TYPE type

  *event type, GR_EVENT_TYPE_TIMER*

- GR_WINDOW_ID wid

  *ID of window timer is destined for.*

- GR_TIMER_ID tid

  *ID of expired timer.*

### 5.14.1 Detailed Description

GR_EVENT_TYPE_TIMER.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.15 GR_EVENT_UPDATE Struct Reference

GR_EVENT_TYPE_UPDATE.

### Data Fields

- GR_EVENT_TYPE type

  *event type*

- GR_WINDOW_ID wid

  *select window id*

- GR_WINDOW_ID subwid

  *update window id (=wid for UPDATE event)*

- GR_COORD x

  *new window x coordinate*

- GR_COORD y

  *new window y coordinate*

- GR_SIZE width

  *new width*

- GR_SIZE height

  *new height*

- GR_UPDATE_TYPE utype

  *update_type*

### 5.15.1 Detailed Description

GR_EVENT_TYPE_UPDATE.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.16 GR_GC_INFO Struct Reference

Graphics context properties returned by the GrGetGCInfo() call.

### Data Fields

- GR_GC_ID gcid

  *GC id (or 0 if no such GC).*

- int mode

  *drawing mode*

- GR_REGION_ID region

  *user region*

- int xoff

  *x offset of user region*

- int yoff

  *y offset of user region*

- GR_FONT_ID font

  *font number*

- GR_COLOR foreground

  *foreground RGB color or pixel value*

- GR_COLOR background

  *background RGB color or pixel value*

- GR_BOOL fgispixelval

  *TRUE if 'foreground' is actually a GR_PIXELVAL.*

- GR_BOOL bgispixelval

  *TRUE if 'background' is actually a GR_PIXELVAL.*

- GR_BOOL usebackground

  *use background in bitmaps*

- GR_BOOL exposure

  *send exposure events on GrCopyArea*

### 5.16.1 Detailed Description

Graphics context properties returned by the GrGetGCInfo() call.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.17 GR_PALETTE Struct Reference

color palette

### Data Fields

- GR_COUNT count

  *# valid entries*

- GR_PALENTRY palette [256]

  *palette*

### 5.17.1 Detailed Description

color palette

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.18 GR_RECT Struct Reference

Nano-X rectangle, different from MWRECT.

### Data Fields

- GR_COORD x

  *upper left x coordinate*

- GR_COORD y

  *upper left y coordinate*

- GR_SIZE width

  *rectangle width*

- GR_SIZE height

  *rectangle height*

### 5.18.1 Detailed Description

Nano-X rectangle, different from MWRECT.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.19 GR_WINDOW_INFO Struct Reference

Window properties returned by the GrGetWindowInfo() call.

### Data Fields

- GR_WINDOW_ID wid

  *window id (or 0 if no such window)*

- GR_WINDOW_ID parent

  *parent window id*

- GR_WINDOW_ID child

  *first child window id (or 0)*

- GR_WINDOW_ID sibling

  *next sibling window id (or 0)*

- GR_BOOL inputonly

  *TRUE if window is input only.*

- GR_BOOL mapped

  *TRUE if window is mapped.*

- GR_BOOL realized

  *TRUE if window is mapped and visible.*

- GR_COORD x

  *parent-relative x position of window*

- GR_COORD y

  *parent-relative y position of window*

- GR_SIZE width

  *width of window*

- GR_SIZE height

  *height of window*

- GR_SIZE bordersize

  *size of border*

- GR_COLOR bordercolor

  *color of border*

- GR_COLOR background

*background color*

- GR_EVENT_MASK eventmask

  *current event mask for this client*

- GR_WM_PROPS props

  *window properties*

- GR_CURSOR_ID cursor

  *cursor id*

- unsigned long processid

  *process id of owner*

### 5.19.1 Detailed Description

Window properties returned by the GrGetWindowInfo() call.

The documentation for this struct was generated from the following file:

- nano-X.h

## 5.20 GR_WM_PROPERTIES Struct Reference

Window manager properties used by the GrGetWMProperties()/GrSetWMProperties() calls.

### Data Fields

- GR_WM_PROPS flags

  *Which properties valid in struct for set.*

- GR_WM_PROPS props

  *Window property bits.*

- GR_CHAR * title

  *Window title.*

- GR_COLOR background

  *Window background color.*

- GR_SIZE bordersize

  *Window border size.*

- GR_COLOR bordercolor

  *Window border color.*

### 5.20.1 Detailed Description

Window manager properties used by the GrGetWMProperties()/GrSetWMProperties() calls.

The documentation for this struct was generated from the following file:

- nano-X.h

# Chapter 6

# Microwindows Nano-X API Page Documentation

## 6.1 Todo List

**Global GrDrawImageFromBuffer(GR_DRAW_ID id, GR_GC_ID gc, GR_COORD x, GR_COORD y, GR_SIZE width**
FIXME document this

**Global GrGetRegionBox(GR_REGION_ID region, GR_RECT ∗rect)** FIXME check Doxygen comments from this point down.

**Global GrLoadImageFromBuffer(void ∗buffer, int size, int flags)** FIXME document this

**Global GrNewPixmap(GR_SIZE width, GR_SIZE height, void ∗pixels)** FIXME Add support for shared memory...

**Global GrReqShmCmds(long shmsize)** FIXME: how does the user decide what size of shared memory area to allocate?

**Global GrSetGCDash(GR_GC_ID gc, char ∗dashes, int count)** FIXME document this

**Global GrSetGCFillMode(GR_GC_ID gc, int fillmode)** FIXME document this

**Global GrSetGCStipple(GR_GC_ID gc, GR_BITMAP ∗bitmap, int width, int height)**
FIXME document this

**Global GrSetGCTile(GR_GC_ID gc, GR_WINDOW_ID pixmap, int width, int height)**
FIXME document this

**Global GrSetGCTSOffset(GR_GC_ID gc, int xoff, int yoff)** FIXME document this

# Index