Nano-X Library API

Gary James
Sr Software Engineer
Critical Link, LLC (http://www.criticallink.com)

gjames@twcny.rr.com
gary.james@criticallink.com

Nano-X Library API

by Gary James

Table of Contents

List of Tables

List of Examples

Release Information

Release Date. May 8, 2001

Nano-X Revision. 0.89-pre7

Chapter 1. Nano-X Function Groups

1.1. General Functions

Function	Description
`GrOpen()`	Open a connection to the nano-X server
`GrClose()`	Close the connection to the nano-X server
`GrFlush()`	Flushes the client/server message buffer
`GrMainLoop()`	Generic application event dispatch loop
`GrGetScreen-Info()`	Return screen information
`GrSetErrorHan-dler()`	Setup an error handler
`GrDefaultEr-rorHandler()`	The default error handler

1.2. Window Functions

Function	Description
`GrNewWindow()`	Create a new window
`GrNewWindowEx()`	Create a new window

Function	Description
`GrNewInputWindow()`	Create a new input window
`GrDestroyWindow()`	Destroy a window
`GrSetWMProperties()`	Set a window's properties
`GrGetWMProperties()`	Retrieve a window's properties
`GrMapWindow()`	Map a window and it's children
`GrUnmapWindow()`	Unmap a window and it's children
`GrRaiseWindow()`	Raise a window
`GrLowerWindow()`	Lower a window
`GrMoveWindow()`	Move a window
`GrResizeWindow()`	Resize a window
`GrReparentWindow()`	Change a window's parent
`GrSetBorderColor()`	Set a window's border color

Function	Description
GrSetWindowBackgroundColor()	Set a window's background color
GrSetWindowBorderColor()	Set a window's border color
GrSetWindowBorderSize()	Set a window's border width
GrSetWindowTitle()	Set a window's title
GrGetWindowInfo()	Retrieve window information
GrSetFocus()	Set the window focus
GrGetFocus()	Get the current focus window
GrNewPixmap()	Create a new pixmap
GrNewPixmapFromData()	Create a new pixmap and initialize it
GrSetBackgroundPixmap()	Set the windows background image
GrClearWindow()	Clear a window
GrCloseWindow()	Close the specified window
GrKillWindow()	Kill the specified window

1.3. Graphics Context Functions

Function	Description
`GrNewGC()`	Create a new graphics context
`GrCopyGC()`	Copy a graphics context into a new graphics context
`GrGetGCInfo()`	Retrieve graphics context settings
`GrDestroyGC()`	Destroy a graphics context
`GrSetGCForeground()`	Change the foreground color of a graphics context
`GrSetGCBackground()`	Change the background color of a graphics context
`GrSetGCUseBackground()`	Enables/disables background usage
`GrSetGCMode()`	Set the drawing mode of a graphics context
`GrSetGCFont()`	Select a font to draw with
`GrGetGCTextSize()`	Calculate size of a text drawing
`GrSetGCRegion()`	Set the clipping region for a graphics context

1.4. Graphics Drawing Functions

Function	Description
`GrPoint()`	Draw a point

Function	Description
GrPoints()	Draw a set of points
GrLine()	Draw a line
GrRect()	Draw a rectangle
GrFillRect()	Draw a filled rectangle
GrPoly()	Draw a polygon
GrFillPoly()	Draw a filled polygon
GrEllipse()	Draw an ellipse or circle
GrFillEllipse()	Draw a filled ellipse or circle
GrArc()	Draw an arc
GrArcAngle()	Draw an arc
GrReadArea()	Read pixel data from a drawable
GrArea()	Draw a pixel array
GrCopyArea()	Copy from one drawable to another
GrBitmap()	Draw a monochrome bitmap
GrFreeImage()	Destroy an image buffer
GrGetImageInfo()	Retrieve information about an image
GrDrawImage-FromFile()	Draw an image from a file
GrLoadImage-FromFile()	Load an image from a file into memory
GrDrawIm-ageToFit()	Draw an image with scaling

Function	Description
`GrDrawImageBits()`	Draw an image
`GrText()`	Draw text
`GrDrawLines()`	Draw a set of lines

1.5. Event Functions

Function	Description
`GrSelectEvents()`	Select event types to receive
`GrGetNextEvent()`	Get an event from the queue
`GrGetNextEventTimeout()`	Get an event from the queue
`GrCheckNextEvent()`	Get an event from the queue
`GrPeekEvent()`	Peek an event from the queue

1.6. Font Functions

Function	Description
GrCreateFont()	Create a font
GrDestroyFont()	Destroy a font
GrSetFontSize()	Set the size of a font
GrSetFontRotation()	Set the angle of a font
GrSetFontAttr()	Change font attributes
GrGetFontInfo()	Get information about a font

1.7. Color Functions

Function	Description
GrGetSystemPalette()	Get the colors of the system palette
GrSetSystemPalette()	Set the colors of the system palette
GrFindColor()	Find closest color match
GrGetSysColor()	Get color by palette index
GR_RGB()	Create a color by RGB components

1.8. Region Functions

Function	Description
GrNewRegion()	Create a new region
GrNewPolygonRegion()	Create a polygon region
GrDestroyRegion()	Destroy a region
GrUnionRectWithRegion()	Form union of rectangle and region
GrUnionRegion()	Form a region from the union of two other regions
GrSubtractRegion()	Form a region from the difference of two regions
GrXorRegion()	Form a region form the XOR two regions
GrIntersectRegion()	Form a region from the intersection of two regions
GrPointInRegion()	Test for point in region
GrRectInRegion()	Test for rectangle in region
GrEmptyRegion()	Test for empty region
GrEqualRegion()	Test two regions for equality

Function	Description
GrOffsetRegion()	Offset a region
GrGetRegionBox()	Get a region's bounding rectangle

1.9. Misc. Functions

Function	Description
GrSetCursor()	Specify a mouse cursor image
GrMoveCursor()	Move the mouse cursor
GrInjectPointerEvent()	Simulate a pointer event
GrInjectKeyboardEvent()	Simulate a keyboard event
GrRegisterInput()	Register a file descriptor to generate events
GrPrepareSelect()	Prepare an fdset for a select
GrServiceSelect()	Dispatch nano-X events
GrReqShmCmds()	Setup a shared memory interface

Function	Description
`GrSetScreen-SaverTimeout()`	Set screen saver timeout
`GrBell()`	Ring bell on server

Chapter 2. Nano-X Function Reference

GrArc()

Name

GrArc() — Draw an arc

Synopsis

```
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 );
```

Description

This function draws an arc on the specified drawable. The arc center is located at the position (*x*,*y*) with a horizontal radius of *rx* and a vertical radius of *ry*. The arc is drawn from the point (*ax*,*ay*) to the point (*bx*,*by*). The endpoints must be specified relative to the arc center and must be positioned on the arc.

> **Note:** If the endpoints are not on the arc, the arc will be drawn incorrectly.

> **Note:** This function does *NOT* use floating point math. Therefore it is very efficient on architectures without floating point processors. If you know you will be running on a floating point processor, then it may be easier to use the

`GrArcAngle()` function. But that function should only be used if you known you have floating point hardware.

Parameters

Type	Name	Description
GR_DRAW_ID	*gc*	The ID of the drawable to draw the arc onto.
GR_GC_ID	*id*	The ID of the graphics context to use when drawing the arc.
GR_COORD	*x*	The X coordinate to draw the arc at relative to the drawable.
GR_COORD	*y*	The Y coordinate to to draw the arc at relative to the drawable.
GR_SIZE	*rx*	The radius of the arc along the X axis.
GR_SIZE	*ry*	The radius of the arc along the Y axis.
GR_COORD	*ax*	The X coordinate start of the arc relative to the drawable.
GR_COORD	*ay*	The Y coordinate start of the arc relative to the drawable.
GR_COORD	*bx*	The X coordinate end of the arc relative to the drawable.
GR_COORD	*by*	The Y coordinate end of the arc relative to the drawable.
int	*type*	The fill style to use when drawing the arc. See below for a list of the available arc types.

Arc Types

The following table lists the possible arc types that may used with this function.

Pixel Format	Description
GR_ARC	Draws just the arc.
GR_ARCOUTLINE	Draws the arc with radius lines from the center to the endpoints. This results in a pie piece shaped outline.
GR_PIE	Draws the arc with radius lines from the center to the endpoints. The resulting pie piece shaped outline is filled with the foreground color.

See Also

GrLine(), GrRect(), GrPoly(), GrEllipse(), GrArcAngle().

GrArcAngle()

Name

GrArcAngle() — Draw an arc

Synopsis

```
void  GrArc ( 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 );
```

Description

This function draws an arc on the specified drawable. The arc center is located at the position (*x*,*y*) with a horizontal radius of *rx* and a vertical radius of *ry*. The arc is drawn from angle *angle1* to *angle2*.

> **Note:** This function requires floating point and is slightly slower than the function `GrArc()`, which does not use floating point.

Parameters

Type	Name	Description
GR_DRAW_ID	*gc*	The ID of the drawable to draw the arc onto.
GR_GC_ID	*id*	The ID of the graphics context to use when drawing the arc.
GR_COORD	*x*	The X coordinate to draw the arc at relative to the drawable.
GR_COORD	*y*	The Y coordinate to to draw the arc at relative to the drawable.
GR_SIZE	*rx*	The radius of the arc along the X axis.
GR_SIZE	*ry*	The radius of the arc along the Y axis.
GR_COORD	*angle1*	The angle of the start of the arc in 1/64 degree increments. A value of 1920 (30*64) will start the arc at 30 degrees.

Type	Name	Description
GR_COORD	*angle2*	The angle of the end of the arc in 1/64 degree increments. A value of 21120 (330*64) will end the arc at -30 degrees.
int	*type*	The fill style to use when drawing the arc. See below for a list of the available arc types.

Arc Types

The following table lists the possible arc types that may used with this function.

Pixel Format	Description
GR_ARC	Draws just the arc.
GR_ARCOUTLINE	Draws the arc with radius lines from the center to the endpoints. This results in a pie piece shaped outline.
GR_PIE	Draws the arc with radius lines from the center to the endpoints. The resulting pie piece shaped outline is filled with the foreground color.

See Also

GrLine(), GrRect(), GrPoly(), GrEllipse(), GrArc().

GrArea()

Name

GrArea() — Draw a pixel array

Synopsis

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

Description

This function draws a bitmap on the specified drawable. The bitmap is drawn at the location (*x*,*y*) relative to the drawable.

The bitmap image is drawn using the specified graphics context. If the graphis context variable `usebackground` is set then pixels in the bitmap that match color with the graphics context's background color are not drawn. This way you can have transparent sections in the bitmap image.

> **Note:** Color conversion is only performed when using the `MWPF_RGB` format.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to red pixel data from.
GR_GC_ID	*gc*	
GR_COORD	*x*	The X coordinate at which to draw the area, relative to the drawable.
GR_COORD	*y*	The Y coordinate at which to draw the area, relative to the drawable.
GR_SIZE	*width*	The width of the area.
GR_SIZE	*height*	The height of the area.
GR_PIXELVAL*	*pixels*	Pointer to an array of pixel data.
int	*pixtype*	The format of the pixel data. See below for a list of the available pixel formats.

Pixel Formats

The following table lists the possible pixel format values that may be used with the `GrArea()` function.

Pixel Format	Description
`MWPF_RGB`	This psuedo format is used as a conversion specifier when working with 32 bit RGB format pixel colors.
`MWPF_PIXELVAL`	This psuedo format is used as a *no conversion specifier when working with GR_PIXELVAL pixel colors.*
`MWPF_PALETTE`	Palettized pixel color format.
`MWPF_TRUECOLOR0888`	Packed 32 bit 0/8/8/8 true color format.
`MWPF_TRUECOLOR888`	Packed 24 bit 8/8/8 truecolor format.
`MWPF_TRUECOLOR565`	Packed 16 bit 5/6/5 truecolor format.

Pixel Format	Description
`MWPF_TRUECOLOR555`	Packed 16 bit 0/5/5/5 truecolor format.
`MWPF_TRUECOLOR332`	Packed 8 bit 3/3/2 truecolor format.

See Also

`GrReadArea()`, `GrCopyArea()`.

`GrBell()`

Name

`GrBell()` — Ring bell on server

Synopsis

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

Description

This function rings the terminal bell on the nano-X server.

GrBitmap()

Name

GrBitmap() — Draw a monochrome bitmap

Synopsis

```
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 );
```

Description

This function fills a rectangular area with a monochrome bitmap. Bits with a value of `1` are drawn in the current foreground color of the specified graphics context. Bits with a value of `0` are drawn in the background color of the graphics context if the `usebackground` flag is set in the GC, otherwise `0` bits have no effect on the drawable. The image data should be an array of aligned 16 bit words.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the bitmap onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the bitmap.

Type	Name	Description
GR_COORD	*x*	The X coordinate to of the rectangular area that the bitmap will be drawn in, relative to the drawable.
GR_COORD	*y*	The Y coordinate to of the rectangular area that the bitmap will be drawn in, relative to the drawable.
GR_SIZE	*width*	The width of the rectangle that the bitmap will be drawn in.
GR_SIZE	*height*	The height of the rectangle that the bitmap will be drawn in.
GR_BITMAP*	*imagebits*	An array of 16 bit words that define the bitmap image.

GrCheckNextEvent()

Name

GrCheckNextEvent() — Get an event from the queue

Synopsis

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

Description

This function retrieves the next nano-X event from the event queue and returns the event in the caller supplied GR_EVENT structure. If the event queue is empty, the function will return immediately with an event type of `GR_EVENT_TYPE_NONE`.

> **Note:** When this function is called all graphics commands queued for the server are flushed.

Parameters

Type	Name	Description
GR_EVENT*	*ep*	Pointer to the caller supplied structure that will receive the next event from the event queue.

See Also

GR_EVENT, `GrSelectEvents()`, `GrGetNextEvent()`, `GrGetNextEventTimeout()`, `GrPeekEvent()`, `GrMainLoop()`.

`GrClearWindow()`

Name

`GrClearWindow()` — Clear a window

Synopsis

```
void  GrClearWindow ( GR_WINDOW_ID  wid ,  GR_BOOL  exposeflag
);
```

Description

This function clears the specified window by setting it to its background color. If the *exposeflag* parameter is non-zero, an exposure event is sent to the window after it has been cleared.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to clear.
GR_BOOL	*exposeflag*	If non-zero an exposure event will be sent to the window after it has been cleared.

See Also

`GrSetWindowBackgroundColor()`, `GrNewWindow()`.

GrClose()

Name

GrClose() — Close the connection to the nano-X server

Synopsis

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

Description

This function closes the connection to the nano-X server and flushes any pending messages in the process.

Example

Example 2-1. Using GrClose()

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
```

```
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
```

```
    }
}
```

See Also

`GrOpen()`

GrCloseWindow()

Name

`GrCloseWindow()` — Close the specified window

Synopsis

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

Description

This function sends a `GR_EVENT_TYPE_CLOSE_REQ` event to the specified window, if the client has selected to receive `GR_EVENT_TYPE_CLOSE_REQ` events on the window. This is used to request an application to shutdown, but not force an immediate shutdown. This way an application can shutdown cleanly, and possibly ask a user if he/she wishes to save files before shutting down.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to close.

See Also

`GR_EVENT_TYPE_CLOSE_REQ`, `GrNewWindow()`, `GrKillWindow()`, `GrDestroyWindow()`, `GrUnmapWindow()`.

GrCopyArea()

Name

`GrCopyArea()` — Copy from one drawable to another

Synopsis

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

Description

This function copies a rectangle of *width* and *height* with upper left corner at coordinates (*srcx*,*srcy*) on the source drawable, *srcid*, to the coordinates (*x*,*y*) on the destination drawable *id*.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the destination drawable.
GR_GC_ID	*gc*	The ID of the graphics context to use when copying the area.
GR_COORD	*x*	The X coordinate of the area, relative to the destination drawable.
GR_COORD	*y*	The Y coordinate of the area, relative to the destination drawable.
GR_SIZE	*width*	The width of the area.
GR_SIZE	*height*	The height of the area.
GR_DRAW_ID	*srcid*	The ID of the source drawable.
GR_COORD	*srcx*	The X coordinate of the area, relative to the source drawable.
GR_COORD	*srcy*	The Y coordinate of the area, relative to the source drawable.
int	*op*	The ROP code for the bit blitter to use when making the copy. In most cases `MWROP_SRCCOPY is a good choice. You can use the ROP codes shown below.`

ROP Codes

The folowing table lists the ROP (Raster OPeration) codes that can be used with the `GrCopyArea()` function.

Table 2-1. ROP Codes

`MWROP_SRCCOPY`	`MWROP_SRCTRANSCOPY`	`MWROP_BLENDCONSTANT`
`MWROP_BLENDFGBG`	`MWROP_BLENDCHANNEL`	`MWROP_STRETCH`
`MWROP_SRCAND`	`MWROP_SRCINVERT`	`MWROP_BLACKNESS`

See Also

`GrReadArea()`, `GrArea()`.

`GrCopyGC()`

Name

`GrCopyGC()` — Copy a graphics context into a new graphics context

Synopsis

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

Description

This function creates a new graphics context and initializes it with values copied from an existing graphics context.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the existing graphics context to copy.

Returns

The ID of the newly created graphics context structure, or 0 if unsuccessful.

See Also

`GrNewGC()`, `GrDestroyGC()`, `GrSetGCForeground()`, `GrSetGCBackground()`, `GrSetGCUseBackground()`, `GrSetGCMode()`, `GrSetGCFont()`, `GrSetGCRegion()`.

`GrCreateFont()`

Name

`GrCreateFont()` — Create a font

Synopsis

```
GR_FONT_ID  GrCreateFont ( GR_CHAR *  name ,  GR_COORD  height ,
 GR_LOGFONT *  plogfont );
```

Description

This function finds the closest available font to the the parameters specified. If *plogfont* is not NULL then the parameters specified in the GR_LOGFONT structure that *plogfont* points to are used to choose the font. Otherwise, if the *height* parameter is non-zero, then the builtin font closest in height to the specified height will be used. Otherwise, the builtin font with a name that matches the *name*, will be chosen. As a lst resort, if none of the previous criteria finds a match, the first builtin font will be returned.

Parameters

Type	Name	Description
GR_CHAR*	*name*	ASCII string containing the face name of the desired font. The table below lists the face names of the builtin fonts.
GR_COORD	*height*	The desired font height.
GR_LOGFONT*	*plogfont*	A pointer to a GR_LOGFONT structure.

Built in font face names for the *name* field.

GR_FONT_SYSTEM_VAR	GR_FONT_OEM_FIXED
GR_FONT_GUI_VAR	GR_FONT_SYSTEM_FIXED

Returns

The ID of the newly created font. This ID must be used in subsequent font functions.

See Also

`GrDestroyFont()`, `GrSetGCFont()`, `GrSetFontSize()`, `GrSetFontRotation()`, `GrSetFontAttr()`, `GrGetFontInfo()`.

`GrDefaultErrorHandler()`

Name

`GrDefaultErrorHandler()` — The default error handler

Synopsis

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

Description

This function is the default error handler that is called if an application specific error handler is NOT installed. This function will print a text error message to `stderr` detailing the error that occured and the function in which the error occured. After printing the message the client application is terminated.

Note: In order to override the default behaviour, an application may define an alternative error handler. Use the `GrSetErrorHandler()` function to specify an alternative error handler function.

Parameters

Type	Name	Description
GR_EVENT*	*ep*	A pointer to a GR_EVENT structure that contains the error that occured.

See Also

`GrSetErrorHandler()`

GrDestroyFont()

Name

`GrDestroyFont()` — Destroy a font

Synopsis

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

Description

This function frees all of the resources allocated to the specified font ID. If the font is not a buitlin font and this is the last reference to the font, the font is unloaded from memory.

Parameters

Type	Name	Description
GR_FONT_ID	*fontid*	The ID of the font to destroy.

See Also

`GrCreateFont()`.

GrDestroyGC()

Name

`GrDestroyGC()` — Destroy a graphics context

Synopsis

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

Description

This function destroys the specified graphics context. Any memory resources allocated to the graphics context are returned to the system.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to destroy.

See Also

`GrNewGC()`, `GrCopyGC()`.

GrDestroyRegion()

Name

`GrDestroyRegion()` — Destroy a region

Synopsis

```
void  GrDestroyRegion ( GR_REGION_ID  region );
```

Description

This function destroys the specified region. Any memory resources allocated to the region are returned to the system.

Parameters

Type	Name	Description
GR_REGION_ID	*region*	The ID of the region to destroy.

See Also

`GrNewRegion()`, `GrNewPolygonRegion()`,

GrDestroyWindow()

Name

`GrDestroyWindow()` — Destroy a window

Synopsis

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

Description

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

> **Note:** This function is also used to destroy pixmaps allocated with the `GrNewPixmap()` function.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to destroy.

See Also

`GrNewWindow()`, `GrNewPixmap()`, `GrNewInputWindow()`, `GrUnmapWindow()`, `GrCloseWindow()`, `GrKillWindow()`.

GrDrawImageBits()

Name

`GrDrawImageBits()` — Draw an image

Synopsis

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

Description

This function draws the specified image onto the specified drawable.

> **Note:** The utility application convbmp that comes with Microwindows will build GR_IMAGE_HDR structures that may be compiled into your application.

> **Note:** The maximum image size that may be sent between the client and server is set by the constant `MAXREQUESTSZ`, defined in the file `microwin/src/nanox/nxproto.h`.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the image onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the image.
GR_COORD	*x*	The X coordinate to draw the image at, relative to the drawable.
GR_COORD	*y*	The Y coordinate to draw the image at, relative to the drawable.

Type	Name	Description
GR_IMAGE_HDR*	*pimage*	A pointer to the image structure.

GrDrawImageFromFile()

Name

GrDrawImageFromFile() — Draw an image from a file

Synopsis

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

Description

This function loads the image file specified by *path* and draws the image at the specified location on the drawable. Supported image types include GIF, JPEG, Windows BMP, PNG, XPM and both ASCII and binary variants of PBM, PGM and PPM.

> **Note:** The actual image types supported by the Nano-X server depend on the image types that were compiled in at microwindows/nano-X library build time.

Note: Filename extensions are irrelevant. The algorithm examines the magic numbers in the file's header to determine the image type.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the image onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the image.
GR_COORD	*x*	The X coordinate to draw the image at.
GR_COORD	*y*	The Y coordinate to draw the image at.
GR_SIZE	*width*	The width to scale the image to.
GR_SIZE	*height*	The height to scale the image to.
char*	*path*	A string containing the filename of the file to load.
int	*flags*	Flags based of the specific image type. Currenly *flags is only used for loading JPEG files. If set to TRUE the JPEG will be loaded in "fast grayscale" mode. If set to FALSE the image will be drawn in RGB color mode.*

See Also

`GrFreeImage()`, `GrGetImageInfo()`, `GrLoadImageFromFile()`, `GrDrawImageToFit()`.

GrDrawImageToFit()

Name

GrDrawImageToFit() — Draw an image with scaling

Synopsis

```
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 );
```

Description

This function draws the image specified by *imageid* on to the drawable *id*. The image is drawn to location (*x*,*y*) on the drawable and the image is stretched to *width* and *height* pixels as it is drawn.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the image onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the image.
GR_COORD	*x*	The X coordinate to draw the image at, relative to the drawable.

Type	Name	Description
GR_COORD	*y*	The Y coordinate to draw the image at, relative to the drawable.
GR_SIZE	*width*	The width to scale the image to.
GR_SIZE	*height*	The height to scale the image to.
GR_IMAGE_ID	*imageid*	The ID of the image to draw.

See Also

`GrFreeImage()`, `GrGetImageInfo()`, `GrDrawImageFromFile()`, `GrLoadImageFromFile()`.

GrDrawLines()

Name

`GrDrawLines()` — Draw a set of lines

Synopsis

```
void  GrDrawLines ( GR_DRAW_ID  id ,  GR_GC_ID  gc ,  GR_POINT *
 points ,  GR_COUNT  count );
```

Description

This function draws a frame polygon on the specified drawable using the graphics context *gc*. The polygon is specified by an array of GR_POINT structures in which each point represents a vertex of the polygon.

> **Note:** This function appears to do the same thing as `GrPoly()`.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the lines onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the lines.
GR_POINT*	*points*	A pointer to an array of GR_POINT structures which define the endpoints of the lines.
GR_COUNT	*count*	The number of points in the point table.

See Also

`GrLine()`, `GrPoly()`.

GrEllipse()

Name

GrEllipse() — Draw an ellipse or circle

Synopsis

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

Description

This function draws an ellipse outline onto the specified drawable. The ellipse is drawn with it's center located at the position (*x*,*y*). The ellipse is drawn with a vertical radius of *ry* and a horizontal radius of *rx*.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the ellipse onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the ellipse.
GR_COORD	*x*	The X coordinate to draw the ellipse at relative to the drawable.

Type	Name	Description
GR_COORD	*y*	The Y coordinate to to draw the ellipse at relative to the drawable.
GR_SIZE	*rx*	The radius of the ellipse along the X axis.
GR_SIZE	*ry*	The radius of the ellipse along the Y axis.

See Also

`GrRect()`, `GrPoly()`, `GrFillEllipse()`, `GrArc()`, `GrArcAngle()`.

GrEmptyRegion()

Name

`GrEmptyRegion()` — Test for empty region

Synopsis

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

Description

This function determines if the specified region is empty.

Parameters

Type	Name	Description
GR_REGION_ID	*region*	The ID of the region to test.

Returns

GR_TRUE if the region is empty, GR_FALSE if the region is *NOT* empty.

See Also

GrPointInRegion(), GrRectInRegion(), GrEqualRegion(),

GrEqualRegion()

Name

GrEqualRegion() — Test two regions for equality

Synopsis

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

Description

This function determines whether the two specified regions are the same.

Parameters

Type	Name	Description
GR_REGION_ID	*rgn1*	The ID of the first of two regions to compare.
GR_REGION_ID	*rgn2*	The ID of the second of two regions to compare.

Returns

`GR_TRUE` if the two regions are equivalent, `GR_FALSE` if the two regions are *NOT* equivalent.

See Also

`GrPointInRegion()`, `GrRectInRegion()`, `GrEmptyRegion()`,

`GrFillEllipse()`

Name

`GrFillEllipse()` — Draw a filled ellipse or circle

Synopsis

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

Description

This function draws a filled ellipse on the drawable *id*. The ellipse is drawn with it's center located at the position (*x*,*y*). The ellipse is drawn with a vertical radius of *ry* and a horizontal radius of *rx*.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the ellipse onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the ellipse.
GR_COORD	*x*	The X coordinate to draw the ellipse at relative to the drawable.
GR_COORD	*y*	The Y coordinate to to draw the ellipse at relative to the drawable.
GR_SIZE	*rx*	The radius of the ellipse along the X axis.
GR_SIZE	*ry*	The radius of the ellipse along the Y axis.

See Also

GrFillRect(), GrFillPoly(), GrEllipse().

GrFillPoly()

Name

GrFillPoly() — Draw a filled polygon

Synopsis

```
void  GrFillPoly ( GR_DRAW_ID  id ,  GR_GC_ID  gc ,  GR_COUNT
count ,  GR_POINT *  pointtable );
```

Description

This function draws a filled polygon on the specified drawable using the graphics context *gc*. The polygon is specified by an array of GR_POINT structures in which each point represents a vertex of the polygon.

> **Note:** The polygon is automatically closed, the last point in the point table does not need to be explicitly specified to be the same as the first point.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the polygon onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the polygon.
GR_COUNT	*count*	The number of points in the point table.
GR_POINT*	*pointtable*	A pointer to an array of GR_POINT structures which define the vertices of the polygon.

See Also

`GrFillRect()`, `GrPoly()`, `GrFillPoly()`, `GrFillEllipse()`.

`GrFillRect()`

Name

`GrFillRect()` — Draw a filled rectangle

Synopsis

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

Description

This function draws a filled rectangle of width (*width*) at position (*x*,*y*) on the specified drawable using the graphics context *gc*.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the rectangle on.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the rectangle.
GR_COORD	*x*	The X coordinate of the rectangle relative to the drawable.
GR_COORD	*y*	The Y coordinate of the rectangle relative to the drawable.
GR_SIZE	*width*	The width of the rectangle in pixels.
GR_SIZE	*height*	The height of the rectangle in pixels.

See Also

`GrRect()`, `GrFillPoly()`, `GrFillEllipse()`.

GrFindColor()

Name

GrFindColor() — Find closest color match

Synopsis

```
void  GrFindColor ( GR_COLOR  color ,  GR_PIXELVAL *  retpixel
);
```

Description

This function calculates the pixel value to use to display the specified color value. The *color* parameter is specified as a architecture independant GR_COLOR, value the pixel value structure is architecture dependent.

Parameters

Type	Name	Description
GR_COLOR	*color*	The color value to find.
GR_PIXELVAL*	*retpixel*	A pointer to the caller supplied GR_PIXELVAL structure.

See Also

GR_PIXELVAL, `GrGetSystemPalette()`, `GrSetSystemPalette()`, `GrGetSysColor()`.

`GrFlush()`

Name

`GrFlush()` — Flushes the client/server message buffer

Synopsis

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

Description

This function flushes the lient to server buffer of any messages that are queued up.

`GrFreeImage()`

Name

`GrFreeImage()` — Destroy an image buffer

Synopsis

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

Description

This function destroys the specified image buffer, and frees the memory that it uses.

Parameters

Type	Name	Description
GR_IMAGE_ID	*id*	The ID of the image buffer.

See Also

`GrGetImageInfo()`, `GrDrawImageFromFile()`, `GrLoadImageFromFile()`, `GrDrawImageToFit()`.

GrGetFocus()

Name

`GrGetFocus()` — Get the current focus window

Synopsis

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

Description

This function gets the ID of the window that currently has the keyboard focus.

Returns

The ID of the window that currently has the keyboard focus.

See Also

`GrSetFocus()`, `GrNewWindow()`.

`GrGetFontInfo()`

Name

`GrGetFontInfo()` — Get information about a font

Synopsis

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

Description

This function retrieves information about the specified font. The font information is returned in the caller supplied GR_FONT_INFO structure.

Parameters

Type	Name	Description
GR_FONT_ID	*fontid*	The ID of the font to retrieve information from.
GR_FONT_INFO*	*fip*	A pointer to the GR_FONT_INFO structure to receive the font information.

See Also

`GrCreateFont()`, `GrSetFontSize()`, `GrSetFontRotation()`, `GrSetFontAttr()`.

GrGetGCInfo()

Name

`GrGetGCInfo()` — Retrieve graphics context settings

Synopsis

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

Description

This function copies information from the GC to the specified GR_GC_INFO structure.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to copy information from.
GR_GC_INFO*	*gcip*	A pointer to a caller supplied GR_GC_INFO structure to receive GC information.

See Also

`GrSetGCForeground()`, `GrSetGCBackground()`, `GrSetGCUseBackground()`, `GrSetGCMode()`, `GrSetGCFont()`, `GrSetGCRegion()`.

GrGetGCTextSize()

Name

`GrGetGCTextSize()` — Calculate size of a text drawing

Synopsis

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

Description

This function calculates the dimensions of the specified text string, if the string were to be drawn with the graphics context *gc*.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to use when calculating the string dimensions.

Type	Name	Description
void*	*str*	The input string. If the string is *NOT zero terminated, then the length of the string must be specified in the* `count parameter.`
int	*count*	The length of the string. This parameter can be set to `-1 if the string is zero terminated and`*flags contains GR_TFASCII.*
int	*flags*	Text rendering flags, can be a combination of the flags listed below.
GR_SIZE*	*retwidth*	Points to the variable that the text width will be returned in.
GR_SIZE*	*retheight*	Points to the variable that the text height will be returned in.
GR_SIZE*	*retbase*	Points to the variable that the text baseline height will be returned in.

The flags parameter is a combination of flags from the following three groups. The combination can include one encoding flag, one alignment flag and multiple attribute flags.

String encoding flags:

`GR_TFASCII`	`GR_TFUTF8`
`GR_TFUC16`	`GR_TFUC32`

Text alignment flags:

`GR_TFTOP`	`GR_TFBASELINE`	`GR_TFBOTTOM`

Text attribute flags:

GR_TFKERNING	GR_TFANTIALIAS	GR_TFUNDERLINE

See Also

GrText(), GrSetGCFont().

GrGetImageInfo()

Name

GrGetImageInfo() — Retrieve information about an image

Synopsis

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

Description

This function returns details of the specified image in the caller supplied GR_IMAGE_INFO structure.

Parameters

Type	Name	Description
GR_IMAGE_ID	*id*	The ID of the image.
GR_IMAGE_INFO*	*iip*	A pointer to an GR_IMAGE_INFO structure to receive information about the image.

See Also

`GrFreeImage()`, `GrDrawImageFromFile()`, `GrLoadImageFromFile()`, `GrDrawImageToFit()`.

`GrGetNextEvent()`

Name

`GrGetNextEvent()` — Get an event from the queue

Synopsis

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

Description

This function retrieves the next nano-X event from the event queue and returns the event in the caller supplied GR_EVENT structure. If the event queue is empty, the function will block until another event occurs.

Parameters

Type	Name	Description
GR_EVENT*	*ep*	Pointer to the caller supplied structure to receive the next event from the event queue.

Example

The following example shows a typical event loop. The first line of the infinite while loop will suspend the client application until an event is available in the event queue. Then the example switches on the event type calling the appropriate application function to process the event.

Example 2-1. Using `GrGetNextEvent()`

```
void typical_event_loop (void)
{
    GR_EVENT  event;

    while (1)
    {
        GrGetNextEvent (&event);
        switch (event.type)
        {
        case GR_EVENT_TYPE_EXPOSURE:
```

```
            process_exposure_event ((GR_EVENT_EXPOSURE*) event);
            break;

        case GR_EVENT_TYPE_BUTTON_DOWN:
            process_button_event ((GR_EVENT_BUTTON*) event);
            break;

        case GR_EVENT_TYPE_KEY_DOWN:
        case GR_EVENT_TYPE_KEY_UP:
            process_key_event ((GR_EVENT_KEYSTROKE*) event);
            break;

        case GR_EVENT_TYPE_SCREENSAVER:
            pro-
cess_screensaver_event ((GR_EVENT_SCREENSAVER*) event);
            break;

        case GR_EVENT_TYPE_CLOSE_REQ:
            GrClose();
            exit (0);
        }
    }
}
```

See Also

`GrSelectEvents()`, `GrGetNextEventTimeout()`, `GrCheckNextEvent()`, `GrPeekEvent()`, `GrMainLoop()`.

GrGetNextEventTimeout()

Name

GrGetNextEventTimeout() — Get an event from the queue

Synopsis

```
void  GrGetNextEventTimeout ( GR_EVENT *  ep ,  GR_TIMEOUT
timeout );
```

Description

This function retrieves the next nano-X event from the event queue and returns the event in the caller supplied GR_EVENT structure. If the event queue is empty, the function will block until either another event occurs or the specified timeout period expires.

> **Note:** If the timeout period expires, a GR_EVENT_TYPE_TIMEOUT event is placed in the GR_EVENT structure pointed to by *ep*.

Parameters

Type	Name	Description

Type	Name	Description
GR_EVENT*	*ep*	Pointer to the caller supplied structure that will receive the next event from the event queue.
GR_TIMEOUT	*timeout*	The timeout period in milliseconds. If `0` is specified the function will block forever, similar to the function `GrGetNextEvent()`.

See Also

`GrSelectEvents()`, `GrGetNextEvent()`, `GrCheckNextEvent()`, `GrPeekEvent()`, `GrMainLoop()`.

GrGetRegionBox()

Name

`GrGetRegionBox()` — Get a region's bounding rectangle

Synopsis

```
int  GrGetRegionBox ( GR_REGION_ID  region ,  GR_RECT *  rect );
```

Description

This function retrieves athe specified regions bounding rectangle. The returned rectangle will enclose all of the region. The function also returns the region's type.

Parameters

Type	Name	Description
GR_REGION_ID	*region*	The ID of the region.
GR_RECT*	*rect*	A caller supplied rectangle to receive the regions bounding rectangle.

Returns

The region type, can be one of the following: `MWREGION_ERROR`, `MWREGION_NULL`, `MWREGION_SIMPLE` or `MWREGION_COMPLEX`.

See Also

`GrNewRegion()`, `GrDestroyRegion()`,

`GrGetScreenInfo()`

Name

`GrGetScreenInfo()` — Return screen properties

Synopsis

```
void  GrGetScreenInfo ( GR_SCREEN_INFO *  sip );
```

Description

This function returns run time information about the screen configuration under which the application is running. The screen properties are returned in the caller supplied GR_SCREEN_INFO structure.

Parameters

Type	Name	Description
GR_SCREEN_INFO	*sip*	A pointer to the caller supplied structure to receive the screen information.

See Also

`GrGetWindowInfo()`,

GrGetSysColor()

Name

`GrGetSysColor()` — Get color by palette index

Synopsis

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

Description

This function returns a color from the system palette that corresponds to the specified palette index.

Parameters

Type	Name	Description
int	*index*	The index into the system color palette table.

Returns

The color at the specified index into the system color table.

See Also

GR_COLOR, `GrGetSystemPalette()`, `GrSetSystemPalette()`, `GrFindColor()`.

GrGetSystemPalette()

Name

`GrGetSystemPalette()` — Get the colors of the system palette

Synopsis

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

Description

This function retrieves the system palete and returns it in the caller supplied GR_PALETTE structure.

Parameters

Type	Name	Description
GR_PALETTE	*pal*	A pointer to the structure to fill the system palette with.

See Also

`GrSetSystemPalette()`, `GrFindColor()`, `GrGetSysColor()`.

GrGetWindowInfo()

Name

GrGetWindowInfo() — Retrieve window information

Synopsis

```
void  GrGetWindowInfo ( GR_WINDOW_ID  wid ,  GR_WINDOW_INFO *
infoptr );
```

Description

This function fills a GR_WINDOW_INFO structure with information regarding the window *wid*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to get info from.
GR_WINDOW_INFO	*infoptr*	A pointer to the caller supplied structure to receive the window information.

See Also

`GrNewWindow()`, `GrMapWindow()`, `GrUnmapWindow()`, `GrRaiseWindow()`, `GrLowerWindow()`, `GrMoveWindow()`, `GrResizeWindow()`, `GrReparentWindow()`, `GrSetBorderColor()`.

GrGetWMProperties()

Name

`GrGetWMProperties()` — Retrieve a window's properties

Synopsis

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

Description

This function returns the window *`wid`*'s window manager properties in the caller supplied GR_WM_PROPERTIES.

> **Note:** It is the callers responsibility to free the *`title`* member of the returned GR_WM_POPERTIES structure, since it is dynamically allocated. The *`title`* will be set to NULL, if the window has no title.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to get the properties of.
GR_WM_PROPERTIES*	*props*	A pointer to the caller supplied GR_WM_PROPERTIES structure that receives the current window properties.

See Also

`GrSetWMProperties()`, `GrNewWindow()`, `GrNewWindowEx()`, `GrGetWindowInfo()`.

GrInjectKeyboardEvent()

Name

`GrInjectKeyboardEvent()` — Simulate a keyboard event

Synopsis

```
void GrInjectKeyboardEvent ( GR_WINDOW_ID wid , GR_KEY keyvalue , GR_KEYMOD modifiers , GR_SCANCODE scancode , GR_BOOL pressed );
```

Description

This function sends a keyboard event to the window *`wid`*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*`wid`*	The ID of the window to send the keyboard event too.
GR_KEY	*`keyvalue`*	The value of the key to inject.
GR_KEYMOD	*`modifiers`*	Key modifier flags.
GR_SCANCODE	*`scancode`*	OEM scancode of the key to inject.
GR_BOOL	*`pressed`*	`GR_TRUE if the injected keyy should appear pressed, GR_FALSE if the injected key should appear released.`

See Also

GR_EVENT_KEYSTROKE, `GrInjectPointerEvent()`.

`GrInjectPointerEvent()`

Name

`GrInjectPointerEvent()` — Simulate a pointer event

Synopsis

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

Description

This function injects a mouse event into the event queue. The mouse event will occur at the coordinates (*x*,*y*). The event will contain the specified button status. The mouse may be hidden or made visible depending on the state of the *visible* parameter.

Note: A `GrFlush()` is performed so that the event takes place immediately.

Parameters

Type	Name	Description
GR_COORD	*x*	The X coordinate to move the cursor too.
GR_COORD	*y*	The Y coordinate to move the cursor too.
int	*button*	The status of the pointer buttons. Indicate which buttons are down by ORing any combination of these flags: `GR_BUTTON_R, GR_BUTTON_M, GR_BUTTON_L.`
int	*visible*	`GR_TRUE to show the mouse cursor, GR_FALSE to hide the cursor.`

See Also

`GrMoveCursor()`, `GrInjectKeyboardEvent()`.

GrIntersectRegion()

Name

`GrIntersectRegion()` — Form a region from the intersection of two regions

Synopsis

```
void  GrIntersectRegion ( GR_REGION_ID  dst_rgn ,  GR_REGION_ID
src_rgn1 ,  GR_REGION_ID  src_rgn2 );
```

Description

This function creates a region from the two specified source regions and places the resulting region in the destination region. The resulting region is the intersection of the two source regions.

Parameters

Type	Name	Description
GR_REGION_ID	*dst_rgn*	The ID of the destination region.

Type	Name	Description
GR_REGION_ID	*src_rgn1*	The ID of the the first of two source regions.
GR_REGION_ID	*src_rgn2*	The ID of the second of two source regions.

See Also

`GrUnionRegion()`, `GrSubtractRegion()`, `GrXorRegion()`, `GrDestroyRegion()`,

GrKillWindow()

Name

`GrKillWindow()` — Kill the specified window

Synopsis

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

Description

This function forces an immediate shutdown of the specified window and disconnects the client that owns the window.

Note: Use this function to kill an application that has locked up and is not responding to `GR_EVENT_TYPE_CLOSE_REQ` events.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to kill.

See Also

`GrNewWindow()`, `GrCloseWindow()`, `GrDestroyWindow()`, `GrUnmapWindow()`.

GrLine()

Name

`GrLine()` — Draw a line

Synopsis

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

Description

This function draws a line on the drawable *id* using the graphics context *gc*, from point (*x1*,*y1*) to point (*x2*,*y2*).

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw a line on.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the line.
GR_COORD	*x1*	The X coordinate of the line's starting point relative to the drawable.
GR_COORD	*y1*	The Y coordinate of the line's starting point relative to the drawable.
GR_COORD	*x2*	The X coordinate of the line's ending point relative to the drawable.
GR_COORD	*y2*	The Y coordinate of the line's ending point relative to the drawable.

See Also

`GrPoint()`, `GrRect()`, `GrPoly()`.

GrLoadImageFromFile()

Name

GrLoadImageFromFile() — Load an image from a file into memory

Synopsis

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

Description

This function loads the image file specified by *path* into a newly created image in the nano-X server's memory and returns the ID of the new image.

Note: The actual image types supported by the Nano-X server depend on the image types that were compiled in at server build time.

Note: Filename extensions are irrelevant. The algorithm examines the magic numbers in the file's header to determine the image type. Supported image types include GIF, JPEG, Windows BMP, PNG, XPM and both ASCII and binary variants of PBM, PGM and PPM.

Note: The file is read from a file by the nano-X server not the nano-X client. This distinction will become more important when nano-X gains support for remote cient operation over network. When the client and server are on the same machine the distinction becomes less important, unless the path is relative. If the

path is relative, it must be specified relative to the servers current working directory rather than the client's.

Parameters

Type	Name	Description
char*	*path*	A string containing the filename of the file to load.
int	*flags*	Flags based of the specific image type. Currenly *flags is only used for loading JPEG files. If set to TRUE the JPEG will be loaded in "fast grayscale" mode. If set to FALSE the image will be drawn in RGB color mode.*

See Also

`GrFreeImage()`, `GrGetImageInfo()`, `GrDrawImageFromFile()`, `GrDrawImageToFit()`.

GrLowerWindow()

Name

`GrLowerWindow()` — Lower a window

Synopsis

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

Description

This function places the specified window at the bottom of its parent's drawing stack, below all of that window's sibling windows.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to lower.

See Also

`GrRaiseWindow()`, `GrNewWindow()`, `GrGetWindowInfo()`.

`GrMainLoop()`

Name

`GrMainLoop()` — Generic application event dispatch loop

Synopsis

```
void  GrMainLoop ( GR_FNCALLBACKEVENT  fncb );
```

Description

This function is the applications main message pump. While this function is running all events that go through the nano-X event queue will be dispatched to the specified event callback function.

> **Note:** This function never returns.

Parameters

Type	Name	Description
GR_FNCALLBACKEVENT	*fncb*	A pointer to the event callback function. This callback function will be called each time an event enters the nano-X event queue.

Example

Example 2-1. Using `GrMainLoop()`

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"
```

```
GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
```

```
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

See Also

GR_FNCALLBACKEVENT, `GrSelectEvents()`, `GrGetNextEvent()`, `GrGetNextEventTimeout()`, `GrCheckNextEvent()`, `GrPeekEvent()`, `GrRegisterInput()`, `GrServiceSelect()`.

`GrMapWindow()`

Name

`GrMapWindow()` — Map a window and it’s children

Synopsis

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

Description

This function recursively maps (shows) the specified window and all of its child windows that 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

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to map.

Example

Example 2-1. Using `GrMapWindow()`

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }
```

```
    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

See Also

`GrNewWindow()`, `GrUnmapWindow()`, `GrGetWindowInfo()`.

`GrMoveCursor()`

Name

`GrMoveCursor()` — Move the mouse cursor

Synopsis

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

Description

This function will move the mouse pointer to the specified coordinates relative to the screen origin ((`0`, `0`) is the upper left cornerof the screen). The cursor's hot spot is positioned at the specified coordinates.

> **Note:** If the mouse is moved over another window, the cursor image will change to the image associated with that window.

Parameters

Type	Name	Description
GR_COORD	*x*	The X coordinate to move the cursor too.
GR_COORD	*y*	The Y coordinate to move the cursor too.

See Also

`GrSetCursor()`.

GrMoveWindow()

Name

`GrMoveWindow()` — Move a window

Synopsis

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

Description

This function moves the upper left corner of window, *wid*, to the coordinates (*x*,*y*) relative to its parent window.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to move.
GR_COORD	*x*	The new X position of the window with respect to its parent window.
GR_COORD	*y*	The new Y position of the window with respect to its parent window.

See Also

`GrNewWindow()`, `GrResizeWindow()`, `GrReparentWindow()`, `GrGetWindowInfo()`.

GrNewGC()

Name

`GrNewGC()` — Create a new graphics context

Synopsis

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

Description

This function creates a new graphics context. The newly created structure is initialized with the default values shown in the following table.

GC Parameter	Default Value
mode	`GR_MODE_SET`
region	No region selected
font	The default system font
foreground	WHITE
background	BLACK
usebackground	`GR_TRUE`

Returns

The ID of the newly created graphics context structure, or `0` if unsuccessful.

See Also

GR_GC_ID, GR_GC_INFO, `GrCopyGC()`, `GrDestroyGC()`, `GrSetGCForeground()`, `GrSetGCBackground()`, `GrSetGCUseBackground()`, `GrSetGCMode()`, `GrSetGCFont()`, `GrSetGCRegion()`, GC Functions.

`GrNewInputWindow()`

Name

`GrNewInputWindow()` — Create a new input window

Synopsis

```
GR_WINDOW_ID  GrNewInputWindow ( GR_WINDOW_ID  parent ,
GR_COORD  x ,  GR_COORD  y ,  GR_COORD  width ,  GR_COORD
height );
```

Description

This function creates a new *input-only* window of *`width`* by *`height`* pixels at coordinates (*`x`*,*`y`*) with respect to the parent window, *`parent`*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*`parent`*	The parent of window of the window that will be created.
GR_COORD	*`x`*	The X position of the new window with respect to its parent window.
GR_COORD	*`y`*	The Y position of the new window with respect to its parent window.
GR_COORD	*`width`*	The width (in pixels) of the new window.
GR_COORD	*`height`*	The hieght (in pixels) of the new window.

Returns

The ID of the newly created window.

See Also

`GrDestroyWindow()`, `GrNewWindow()`, `GrNewWindowEx()`.

GrNewPixmap()

Name

`GrNewPixmap()` — Create a new pixmap

Synopsis

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

Description

This function creates a new server side pixmap of the specified width and height.

> **Note:** A pixmap is basically an offscreen window. You can draw to a pixmap just as you would an on screen window. The return value of this function is a bit misleading, in that it should probably be a GR_DRAW_ID rather than a window ID. From a drawing point of view nano-X does not make much of a difference between window IDs and drawable IDs.

Note: When finished with the pixmap, the application must free the resources allocated to the pixmap by calling the function `GrDestroyWindow()` with the pixmap ID as the ID of the window to destroy.

Parameters

Type	Name	Description
GR_SIZE	*`width`*	The width in pixels of the pixmap.
GR_SIZE	*`height`*	The height in pixels of the pixmap.
void*	*`addr`*	Optional pointer to pixel buffer. If you have a pixel buffer already allocated for the pixmap, then pass that buffer via this parameter. If this parameter is `NULL then the function will attemp to allocate a proper sized pixel buffer. If the pixel buffer is automatically allocated the buffer will automatically be freed when the pixmap is destroyed using GrDestroyWindow().`

Returns

The ID of the newly created pixmap.

See Also

`GrNewPixmapFromData()`, `GrDestroyWindow()`, `GrCopyArea()`,

GrSetBackgroundPixmap().

GrNewPixmapFromData()

Name

GrNewPixmapFromData() — Create a new pixmap and initialize it

Synopsis

```
GR_WINDOW_ID  GrNewPixmapFromData ( GR_SIZE  width ,  GR_SIZE
height ,  GR_COLOR  foreground ,  GR_COLOR  background ,  void *
 bits ,  int  flags );
```

Description

This function creates a new server side pixmap of the specified width and height. The pixmap is initialized with a monochrome bitmap corresponding to the specified bit array. If specified byte and/or bit reversal will be performed on each short word within the bit array.

> **Note:** A pixmap is basically an offscreen window. You can draw to a pixmap just as you would an on screen window. The return value of this function is a bit misleading, in that it should probably be a GR_DRAW_ID rather than a window ID. From a drawing point of view nano-X does not make much of a difference between window IDs and drawable IDs.

Note: When finished with the pixmap, the application must free the resources allocated to the pixmap by calling the function `GrDestroyWindow()` with the pixmap ID as the ID of the window to destroy.

Parameters

Type	Name	Description
GR_SIZE	*`width`*	The width in pixels of the pixmap.
GR_SIZE	*`height`*	The height in pixels of the pixmap.
GR_COLOR	*`foreground`*	The color to use as the pixmap foreground.
GR_COLOR	*`background`*	The color to use as the pixmap background.
void*	*`bits`*	Pinter to the bit array. All set bits in this array are drawn in the specfied foreground color. All clear bits in this array are drawn in the specified background color.
int	*`flags`*	This fields hold flags from the table below.

Mode	Description
`GR_BMDATA_BYTEREVERSE`	Swap every other byte of the bit array (short word byte swap).
`GR_BMDATA_BYTESWAP`	Reverse the bit order of every byte within the bit array.

Returns

The ID of the newly created pixmap.

See Also

`GrNewPixmap()`, `GrDestroyWindow()`, `GrCopyArea()`, `GrSetBackgroundPixmap()`.

GrNewPolygonRegion()

Name

`GrNewPolygonRegion()` — Create a polygon region

Synopsis

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

Description

This function creates a new region and returns its ID. The new region is created by connecting the points in the specified GR_POINT array.

Parameters

Type	Name	Description

Type	Name	Description
int	*mode*	The method for handling overlapping sections of the polygon. See the mode description table below.
GR_COUNT	*count*	The number of points in the GR_POINT array used to define the polygon.
GR_POINT*	*points*	A pointer to an array of GR_POINT structures that define the vertices of the polygon.

Mode	Description
MWPOLY_EVENODD	Areas of the polygon that overlap an odd number of times are not a part of the resulting region.
MWPOLY_WINDING	Areas of the polygon that overlap are always considered a part of the region.

Returns

The ID of the newly created region.

See Also

GR_POINT, GrNewRegion(), GrDestroyRegion(), GrPoly().

GrNewRegion()

Name

GrNewRegion() — Create a new region

Synopsis

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

Description

This function creates a new region and returns the ID of the newly created region. The new region is created empty, i.e. the region has no width or height.

Returns

The ID of the newly created region.

See Also

GrNewPolygonRegion(), GrDestroyRegion(), Region Functions.

GrNewWindow()

Name

GrNewWindow() — Create a new window

Synopsis

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

Description

This function will create a new window with the specified parent window and the specified window attributes.

> **Note:** This function has been depreciated. You should use GrNewWindowEx() instead since it works better in the presence of a window manager.

Parameters

Type	Name	Description
GR_WINDOW_ID	*parent*	The parent of window of the window that will be created.

Type	Name	Description
GR_COORD	*x*	The X position of the new window with respect to its parent window.
GR_COORD	*y*	The Y position of the new window with respect to its parent window.
GR_COORD	*width*	The width (in pixels) of the new window.
GR_COORD	*height*	The hieght (in pixels) of the new window.
GR_COORD	*bordersize*	The width in pixels of the window border.
GR_COLOR	*background*	The color of the window background.
GR_COLOR	*bordercolor*	he color of the window border.

Returns

The ID of the newly created window or `0` in case of failure.

See Also

`GrDestroyWindow()`, `GrNewWindowEx()`, `GrNewInputWindow()`, `GrMapWindow()`, `GrRaiseWindow()`, `GrMoveWindow()`, `GrResizeWindow()`, `GrReparentWindow()`, `GrSetBorderColor()`, `GrClearWindow()`, `GrGetWindowInfo()`, Window Functions.

`GrNewWindowEx()`

Name

`GrNewWindowEx()` — Create a new window

Synopsis

```
GR_WINDOW_ID  GrNewWindowEx ( GR_WM_PROPS  props ,  GR_CHAR *
title ,  GR_WINDOW_ID  parent ,  GR_COORD  x ,  GR_COORD  y ,
GR_COORD  width ,  GR_COORD  height ,  GR_COLOR  background );
```

Description

This function will create a new window with the specified parent window and the specified window attributes and properties.

Parameters

Type	Name	Description
GR_WM_PROPS	*props*	Window manager properties for this window.
GR_CHAR	*title*	The text that will appear in the title bar of this window if the window is a top level window.
GR_WINDOW_ID	*parent*	The parent of window of the window that will be created.
GR_COORD	*x*	The X position of the new window with respect to its parent window.
GR_COORD	*y*	The Y position of the new window with respect to its parent window.

Type	Name	Description
GR_COORD	*width*	The width (in pixels) of the new window.
GR_COORD	*height*	The hieght (in pixels) of the new window.
GR_COLOR	*background*	The background color of this window.

Returns

The ID of the newly created window or `0` in case of failure.

Example

Example 2-1. Using `GrNewWindowEx()`

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
```

```
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

See Also

GR_WM_PROPS, `GrDestroyWindow()`, `GrNewWindow()`, `GrNewInputWindow()`, `GrSetWMProperties()`, `GrGetWMProperties()`, Window Functions.

`GrOffsetRegion()`

Name

`GrOffsetRegion()` — Offset a region

Synopsis

```
void  GrOffsetRegion ( GR_REGION_ID  region ,  GR_SIZE  dx ,
GR_SIZE  dy );
```

Description

This function offsets the specified region by *`dx`* along the X axis and by *`dy`* along the Y axis.

Parameters

Type	Name	Description
GR_REGION_ID	*`region`*	The ID of the region to offset.

Type	Name	Description
GR_SIZE	*dx*	The X distance, in pixels, to offset the region.
GR_SIZE	*dy*	The Y distance, in pixels, to offset the region.

See Also

`GrNewRegion()`, `GrDestroyRegion()`,

`GrOpen()`

Name

`GrOpen()` — Open a connection to the nano-X server

Synopsis

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

Description

This function opens a connection to the graphics server. This must be the first nano-X function called by your application.

Returns

The file descriptor `fd` of the connection to the server, or -1 in case of an error.

Example

Example 2-1. Using `GrOpen()`

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);
```

```
    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

See Also

`GrClose()`

GrPeekEvent()

Name

`GrPeekEvent()` — Peek an event from the queue

Synopsis

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

Description

This function retrieves the next nano-X event from the event queue without actually removing the event from the queue. The retrieved event is returned in the caller supplied GR_EVENT structure. If the event queue is empty, the function will return with an event type of `GR_EVENT_TYPE_NONE`.

Parameters

Type	Name	Description
GR_EVENT*	*ep*	Pointer to the caller supplied structure to receive the next event from the event queue.

Returns

`1` if an event is returned from the queue, `0` if the queue was empty.

See Also

GrSelectEvents(), GrGetNextEvent(), GrGetNextEventTimeout(), GrCheckNextEvent(), GrMainLoop().

GrPoint()

Name

GrPoint() — Draw a point

Synopsis

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

Description

This function draws a single point on the specified drawable at the coordinates (*x*,*y*) using the specified graphics context.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the point on.

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the point.
GR_COORD	*x*	The X coordinate of the point relative to the drawable.
GR_COORD	*y*	The Y coordinate of the point relative to the drawable.

See Also

`GrPoints()`, `GrLine()`, `GrRect()`, `GrPoly()`, `GrEllipse()`.

`GrPointInRegion()`

Name

`GrPointInRegion()` — Test for point in region

Synopsis

```
GR_BOOL  GrPointInRegion ( GR_REGION_ID  region ,  GR_COORD  x ,
 GR_COORD  y );
```

Description

This function tests to see if the specified point (*x*, *y*) is within the specified region.

Parameters

Type	Name	Description
GR_REGION_ID	*region*	The ID of the region to test for inclusion of the point.
GR_COORD	*x*	The X coordinate of the point to test.
GR_COORD	*y*	The Y coordinate of the point to test.

Returns

`GR_TRUE` if the point is within the region, `GR_FALSE` if the point is outside the region.

See Also

`GrRectInRegion()`, `GrEmptyRegion()`, `GrEqualRegion()`,

`GrPoints()`

Name

`GrPoints()` — Draw a set of points

Synopsis

```
void  GrPoints ( GR_DRAW_ID  id ,  GR_GC_ID  gc ,  GR_COUNT  count ,  GR_POINT *  pointtable );
```

Description

This function draws a set of points defined by the specified point table onto the specified drawable.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the points onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the points.
GR_COUNT	*count*	The number of points in the point table.
GR_POINT*	*pointtable*	A pointer to an array of GR_POINT structures which list the points to draw.

See Also

`GrPoint()`, `GrLine()`, `GrRect()`, `GrPoly()`.

GrPoly()

Name

GrPoly() — Draw a polygon

Synopsis

```
void  GrPoly ( GR_DRAW_ID  id ,  GR_GC_ID  gc ,  GR_COUNT  count
,  GR_POINT *  pointtable );
```

Description

This function draws a frame polygon on the specified drawable using the graphics context *gc*. The polygon is specified by an array of GR_POINT structures in which each point represents a vertex of the polygon.

> **Note:** The polygon is *NOT* automatically closed. If you wish to draw a closed polygon the first and last point in the point table must specify the same coordinates.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the polygon onto.

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the polygon.
GR_COUNT	*count*	The number of points in the point table.
GR_POINT*	*pointtable*	A pointer to an array of GR_POINT structures which define the vertices of the polygon.

See Also

`GrFillPoly()`, `GrDrawLines()`, `GrRect()`, `GrEllipse()`, `GrArc()`, `GrArcAngle()`.

GrPrepareSelect()

Name

`GrPrepareSelect()` — Prepare an fdset for a select

Synopsis

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

Description

This function prepares a file descriptor set for use in a subsequent `select()` call. The file descriptor set is initialized with the nano-X client/server socket descriptor and all registered external file descriptors. The parameter *`maxfd`* is set to the highest of the file descriptors in the set.

Parameters

Type	Name	Description
int*	*`maxfd`*	Pointer to an integer to receive the highest file descriptor in the file descriptor set.
void*	*`rfdset`*	A pointer to a caller supplied file descriptor set structure to fill in.

See Also

`GrServiceSelect()`, `GrRegisterInput()`, `GrMainLoop()`.

`GrRaiseWindow()`

Name

`GrRaiseWindow()` — Raise a window

Synopsis

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

Description

This function places the specified window at the top of its parent's drawing stack, above all of the window's siblings.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to raise.

See Also

`GrLowerWindow()`, `GrNewWindow()`, `GrMapWindow()`, `GrGetWindowInfo()`.

GrReadArea()

Name

`GrReadArea()` — Read pixel data from a drawable

Synopsis

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

Description

This function reads a rectangle of pixel data from the specified drawable into the caller supplied pixel buffer. The pixel data is read from a rectangular region at position (*x*, *y*) of size (*width*, *height*).

> **Note:** If the drawable is a window, then the pixel data returned will be pixel values from the appropiate position on the screen. If another window covers the specified window, then the visible window's image will be returned. If the window *wid* is unmapped or partially outside a window boundary, black pixels will be returned in the nonvisible section of the area.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to red pixel data from.
GR_COORD	*x*	The X coordinate of the read rectangle, relative to the drawable.
GR_COORD	*y*	The Y coordinate of the read rectangle, relative to the drawable.
GR_SIZE	*width*	The width of the read rectangle, relative to the drawable.

Type	Name	Description
GR_SIZE	*height*	The height of the read rectangle, relative to the drawable.
GR_PIXELVAL*	*pixels*	Pointer to a caller supplied area of memory to read the pixel data into.

See Also

`GrArea()`, `GrCopyArea()`.

GrRect()

Name

`GrRect()` — Draw a rectangle

Synopsis

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

Description

This function draws a frame rectangle of width (*width*) and height (*height*) at position (*x*,*y*) on the specified drawable using the graphics context *gc*.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the rectangle on.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the rectangle.
GR_COORD	*x*	The X coordinate of the rectangle relative to the drawable.
GR_COORD	*y*	The Y coordinate of the rectangle relative to the drawable.
GR_SIZE	*width*	The width of the rectangle in pixels.
GR_SIZE	*height*	The height of the rectangle in pixels.

See Also

`GrLine()`, `GrFillRect()`, `GrPoly()`, `GrEllipse()`.

`GrRectInRegion()`

Name

`GrRectInRegion()` — Test for rectangle in region

Synopsis

```
int GrRectInRegion ( GR_REGION_ID region , GR_COORD x ,
GR_COORD y , GR_COORD width , GR_COORD height );
```

Description

This function tests to see if the specified rectangle (*x*, *y*, *width*, *height*) is contained within (or partially contained within) the specified region.

Parameters

Type	Name	Description
GR_REGION_ID	*region*	The ID of the region to test.
GR_COORD	*x*	The X coordinate of the rectangle.
GR_COORD	*y*	The Y coordinate of the rectangle.
GR_COORD	*w*	The width of the rectangle in pixels.
GR_COORD	*h*	The height of the rectangle in pixels.

Returns

This function returns on of the following values:

Value	Description
GR_RECT_OUT	If the rectangle is completely outside the region.
GR_RECT_ALLIN	If the rectangle is completely inside the region.

Value	Description
GR_RECT_PARTIN	If the rectangle is partially within the region.

See Also

GrPointInRegion(), GrEmptyRegion(), GrEqualRegion(),

GrRegisterInput()

Name

GrRegisterInput() — Register a file descriptor to generate events

Synopsis

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

Description

This function allows you to register additional file descriptors to monitor in the main select loop of the nano-X application. A GR_EVENT_FDINPUT event will be sent through the nano-X event queue when the specified file descriptor has data available to be read.

Parameters

Type	Name	Description
int	*fd*	The file descriptor to monitor.

See Also

`GrPrepareSelect()`, `GrServiceSelect()`, GR_EVENT_FDINPUT,

`GrReparentWindow()`

Name

`GrReparentWindow()` — Change a window's parent

Synopsis

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

Description

This function changes the parent window of the window *wid* to the specified new parent window. It places the window at the coordinates (*x*,*y*) relative to the new parent window.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to reparent.
GR_WINDOW_ID	*pwid*	The ID of the new parent window.
GR_COORD	*x*	The new X position of the window with respect to its new parent window.
GR_COORD	*y*	The new Y position of the window with respect to its new parent window.

See Also

`GrNewWindow()`, `GrMoveWindow()`, `GrGetWindowInfo()`.

`GrReqShmCmds()`

Name

`GrReqShmCmds()` — Setup a shared memory interface

Synopsis

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

Description

This function requests a shared memory area for the use of transferring command arguments between a nano-X client and nano-X server. Generally nano-X uses socket calls to transfer command arguments, but using shared memory can increase system performance. The use of shared memory or sockets is transparent to the application programmer aside from this function call.

> **Note:** It is safe to call this function if shared emory support is not compiled into your nano-X library, because nano-X will transparently default to use the socket interface. Nano-X will also transparently roll over to using the socket interface if the shared memory allocation fails.

Parameters

Type	Name	Description
long	*shmsize*	The size in bytes of the shared memory buffer.

GrResizeWindow()

Name

`GrResizeWindow()` — Resize a window

Synopsis

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

Description

This function resizes the specified window to the specified width and height.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to resize.
GR_SIZE	*width*	The new width (in pixels) of the window.
GR_SIZE	*height*	The new height (in pixels) of the window.

See Also

`GrNewWindow()`, `GrMoveWindow()`, `GrGetWindowInfo()`.

GR_RGB()

Name

`GR_RGB()` — Create a color by RGB components

Synopsis

```
GR_COLOR  GR_RGB ( unsigned char  r ,  unsigned char  g ,
unsigned char  b );
```

Description

This macro constructs a GR_COLOR variable type from it's component colors. Each parameter defines the level of the component (red, green and blue) colors.

Parameters

Type	Name	Description
unsigned char	*r*	The level of RED component in the resulting color.
unsigned char	*g*	The level of GREEN component in the resulting color.
unsigned char	*b*	The level of BLUE component in the resulting color.

Returns

A GR_COLOR value.

See Also

GR_COLOR.

GrSelectEvents()

Name

GrSelectEvents() — Select event types to receive

Synopsis

```
void  GrSelectEvents ( GR_WINDOW_ID  wid ,  GR_EVENT_MASK  eventmask );
```

Description

This function allows you to select the event types which you want returned from the specified window. The *event_mask* can be a bitwise OR of multiple events.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to receive events on.

Type	Name	Description
GR_EVENT_MASK	*eventmask*	A bitmask that specifies the events that should be sent from the window.

Example

Example 2-1. Using `GrSelectEvents()`

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
```

```
                        GR_ROOT_WINDOW_ID,
                        50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                        GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

See Also

GrGetNextEvent(), GrGetNextEventTimeout(), GrCheckNextEvent(), GrPeekEvent(), GrMainLoop().

GrServiceSelect()

Name

`GrServiceSelect()` — Dispatch nano-X events

Synopsis

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

Description

This function is used by `GrMainLoop()` to dispatch events to its event callback function. You can use this function if you intend to roll your own event dispatcher rather than to use `GrMainLoop()`.

Parameters

Type	Name	Description
void*	*rfdset*	A pointer to the file descriptor set that select received.
GR_FNCALLBACKEVENT	*fncb*	A pointer to the event callback function. This callback function will be called each time an event enters the nano-X event queue.

See Also

GR_FNCALLBACKEVENT, `GrPrepareSelect()`, `GrMainLoop()`.

`GrSetBackgroundPixmap()`

Name

`GrSetBackgroundPixmap()` — Set the windows background image

Synopsis

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

Description

This function sets the background of the window *`wid`* to display the image from *`pixmap`*. The *`flags`* parameter specifies how to draw the pixmap (centered, top-left, tiled, etc.). If the *`pixmap`* parameter is zero, the background pixmap will be disabled for the window, and the window will revert to using a solid color fill.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to set the background pixmap for.
GR_WINDOW_ID	*pixmap*	ID of the pixmap to use as a background image.
int	*flags*	The pixmap drawing flags. These flags define how the pixmap should be drawn within the window. See the table below.

Table 2-1. GrSetBackgroundPixmap Flags

Flag	Description
GR_BACKGROUND_TILE	Tile the pixmap images across the window.
GR_BACKGROUND_CENTER	Center the pixmap image in the window.
GR_BACKGROUND_TOPLEFT	Draw the pixmap image in the upper left corner of the window.
GR_BACKGROUND_STRETCH	Stretch the pixmap image to fit the window.
GR_BACKGROUND_TRANS	Don’t fill in the gaps in the window.

See Also

GrNewWindow(), GrNewPixmap(), GrDrawImageToFit().

GrSetBorderColor()

Name

GrSetBorderColor() — Set a window's border color

Synopsis

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

Description

This function sets the border color of the specified window to *color*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to set the border color for.
GR_COLOR	*color*	The new border color for the window.

See Also

GrSetWindowBorderColor(), GrNewWindow(), GrGetWindowInfo().

GrSetCursor()

Name

GrSetCursor() — Specify a mouse cursor image

Synopsis

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

Description

This function allows you to specify the image to use as the mouse pointer for the specified window and it's children.

> **Note:** Pixels that are not set in either the foreground or the background bitmaps will be transparent.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to set the cursor for.

Type	Name	Description
GR_SIZE	*width*	The width of the pointer bitmap in pixels.
GR_SIZE	*height*	The height of the pointer bitmap in pixels.
GR_COORD	*hotx*	The X coordinate within the bitmap used as the target for the pointer.
GR_COORD	*hoty*	The Y coordinate within the bitmap used as the target for the pointer.
GR_COLOR	*foreground*	The color to use for the foreground bitmap image.
GR_COLOR	*background*	The color to use for the background bitmap image.
GR_BITMAP*	*fgbitmap*	Pointer to a bitmap data array to use as the foreground bitmap.
GR_BITMAP*	*bgbitmap*	Pointer to a bitmap data array to use as the background bitmap.

Example

Example 2-1. Using `GrSetCursor()`

```
void set_x_cursor (GR_WINDOW_ID wid)
{
    GR_BITMAP fg_bitmap[16];
    GR_BITMAP bg_bitmap[16];

    fg_bitmap[0]  = 0x8001;  /* X______________X */
    fg_bitmap[1]  = 0x4002;  /* _X____________X_ */
    fg_bitmap[2]  = 0x2004;  /* __X__________X__ */
    fg_bitmap[3]  = 0x1008;  /* ___X________X___ */
    fg_bitmap[4]  = 0x0810;  /* ____X______X____ */
    fg_bitmap[5]  = 0x0420;  /* _____X____X_____ */
```

```
    fg_bitmap[6]  = 0x0240;  /* ______X__X______ */
    fg_bitmap[7]  = 0x0180;  /* _______XX_______ */
    fg_bitmap[8]  = 0x0180;  /* _______XX_______ */
    fg_bitmap[9]  = 0x0240;  /* ______X__X______ */
    fg_bitmap[10] = 0x0420;  /* _____X____X_____ */
    fg_bitmap[11] = 0x0810;  /* ____X______X____ */
    fg_bitmap[12] = 0x1008;  /* ___X________X___ */
    fg_bitmap[13] = 0x2004;  /* __X__________X__ */
    fg_bitmap[14] = 0x4002;  /* _X____________X_ */
    fg_bitmap[15] = 0x8001;  /* X______________X */

    bg_bitmap[0]  = 0x4002;  /* _X____________X_ */
    bg_bitmap[1]  = 0xA005;  /* X_X__________X_X */
    bg_bitmap[2]  = 0x500A;  /* _X_X________X_X_ */
    bg_bitmap[3]  = 0x2814;  /* __X_X______X_X__ */
    bg_bitmap[4]  = 0x1428;  /* ___X_X____X_X___ */
    bg_bitmap[5]  = 0x0A50;  /* ____X_X__X_X____ */
    bg_bitmap[6]  = 0x05A0;  /* _____X_XX_X_____ */
    bg_bitmap[7]  = 0x0240;  /* ______X__X______ */
    bg_bitmap[8]  = 0x0240;  /* ______X__X______ */
    bg_bitmap[9]  = 0x05A0;  /* _____X_XX_X_____ */
    bg_bitmap[10] = 0x0A50;  /* ____X_X__X_X____ */
    bg_bitmap[11] = 0x1428;  /* ___X_X____X_X___ */
    bg_bitmap[12] = 0x2814;  /* __X_X______X_X__ */
    bg_bitmap[13] = 0x500A;  /* _X_X________X_X_ */
    bg_bitmap[14] = 0xA005;  /* X_X__________X_X */
    bg_bitmap[15] = 0x4002;  /* _X____________X_ */

    GrSetCursor (wid, 16, 16, 8, 8,
                 BLACK, BLACK, fg_bitmap, bg_bitmap);
}
```

See Also

`GrMoveCursor()`.

GrSetErrorHandler()

Name

GrSetErrorHandler() — Setup an error handler

Synopsis

```
GR_FNCALLBACKEVENT  GrSetErrorHandler ( GR_FNCALLBACKEVENT  fncb
);
```

Description

This function allows you to specify an error handing function, for all errors that the server sends to the client. If an error occurs the specified error handler is called with an error event. If a NULL function pointer is specified, then errors will be sent through the nano-X event queue rather than through an error handler callback function.

Parameters

Type	Name	Description
GR_FNCALLBACKEVENT	*fncb*	A pointer to the error handler function or NULL to send errors through the event queue.

Returns

A pointer to the previous error handler function.

See Also

`GrDefaultErrorHandler()`, GR_ERROR, GR_EVENT_ERROR.

`GrSetFocus()`

Name

`GrSetFocus()` — Set the window focus

Synopsis

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

Description

This function sets the keyboard focus to the specified window.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to set keyboard focus to.

See Also

`GrGetFocus()`, `GrNewWindow()`.

GrSetFontAttr()

Name

`GrSetFontAttr()` — Change font attributes

Synopsis

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

Description

This function changes the attributes of the specified font.

Parameters

Type	Name	Description
GR_FONT_ID	*fontid*	The ID of the font in which the attributes will be modified.
int	*setflags*	A bitwise OR of all the font attribute flags to set.
int	*clrflags*	A bitwise OR of all the font attribute flags to clear.

Table 2-1. Font Attribute Flags

GR_TFKERNING	GR_TFANTIALIAS	GR_TFUNDERLINE

See Also

GrCreateFont(), GrSetFontSize(), GrSetFontRotation(), GrGetFontInfo().

GrSetFontRotation()

Name

GrSetFontRotation() — Set the angle of a font

Synopsis

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

Description

This function changes the rotation of the specified font.

Parameters

Type	Name	Description
GR_FONT_ID	*fontid*	The ID of the font to change the rotation of.
int	*tenthsdegrees*	The new angle of rotation for the font.

See Also

`GrCreateFont()`, `GrSetFontSize()`, `GrSetFontAttr()`, `GrGetFontInfo()`.

GrSetFontSize()

Name

`GrSetFontSize()` — Set the size of a font

Synopsis

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

Description

This function changes the size of the specified font.

Parameters

Type	Name	Description
GR_FONT_ID	*fontid*	The ID of the font to change the size of.
GR_COORD	*size*	The new size for the font.

See Also

`GrCreateFont()`, `GrSetFontRotation()`, `GrSetFontAttr()`, `GrGetFontInfo()`.

GrSetGCBackground()

Name

GrSetGCBackground() — Change the background color of a graphics context

Synopsis

```
void  GrSetGCBackground ( GR_GC_ID  gc ,  GR_COLOR  background
);
```

Description

This function changes the background color of the specified graphics context to the specified color.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to modify.
GR_COLOR	*background*	The new background color for the graphics context.

See Also

GrNewGC(), GrGetGCInfo(), GrSetGCForeground(),

`GrSetGCUseBackground()`.

GrSetGCFont()

Name

`GrSetGCFont()` — Select a font to draw with

Synopsis

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

Description

This function sets the font for the specified graphics context.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to modify.
GR_FONT_ID	*font*	The ID of the new font to use when drawing text with the GC.

See Also

`GrNewGC()`, `GrGetGCInfo()`.

`GrSetGCForeground()`

Name

`GrSetGCForeground()` — Change the foreground color of a graphics context

Synopsis

```
void  GrSetGCForeground ( GR_GC_ID  gc ,  GR_COLOR  foreground
);
```

Description

This function changes the foreground color of the specified graphics context to the specified color.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to modify.

Type	Name	Description
GR_COLOR	*foreground*	The new foreground color for the graphics context.

See Also

`GrNewGC()`, `GrGetGCInfo()`, `GrSetGCBackground()`, `GrSetGCMode()`.

GrSetGCMode()

Name

`GrSetGCMode()` — Set the drawing mode of a graphics context

Synopsis

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

Description

This function sets the drawing mode for the specified graphics context. The mode defines how nano-X will draw pixels over each other. Generally drawi9ng is done with the `GR_MODE_SET`. In this case if you draw a black object you get a black object. In the other drawing modes you would get a loical combination of the black object and whatever else is already on the screen.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to modify.
int	*mode*	The new drawing mode.

The following table shows the drawing modes that are available for use with graphics contexts in nano-X.

Table 2-1. Drawing Modes

Value	Description
GR_MODE_SET	When drawing the graphic output will represent the selected forground color.
GR_MODE_XOR	When drawing the graphic output will be th XOR of the GC's foreground color and the current color on the drawable.
GR_MODE_OR	When drawing the graphic output will be th OR of the GC's foreground color and the current color on the drawable.
GR_MODE_AND	When drawing the graphic output will be th AND of the GC's foreground color and the current color on the drawable.
GR_MODE_DRAWMASK	Set bits in this mask correspond to GC mode bits that define drawing style. Clear bits of this mask correspond to GC mode bits that have an extended meaning beyond the drawing style. In this table all of the preceding mode bits define drawing style, all of the following bits have an extended meaning.

Value	Description
GR_MODE_EXCLUDECHILDREN	If this flag is set, then while clipping child windows are excluded from the clip region. Normally the area covered by child windows is clipped when drawing on the parent window. This flag disables the normal clipping action.

See Also

GrNewGC(), GrGetGCInfo().

GrSetGCRegion()

Name

GrSetGCRegion() — Set the clipping region for a graphics context

Synopsis

```
void  GrSetGCRegion ( GR_GC_ID  gc ,  GR_REGION_ID  region );
```

Description

This function sets the clipping region for the specified graphics context to the specified region. Subsequent drawing functions will not draw beyond the limits of the clipping region.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to set the clipping region of.
GR_REGION_ID	*region*	The ID of the region to use, 0 to set no clipping region.

See Also

`GrNewRegion()`, `GrNewGC()`, `GrGetGCInfo()`.

GrSetGCUseBackground()

Name

`GrSetGCUseBackground()` — Enables/disables background usage

Synopsis

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

Description

This function sets the use background flag in the specified graphics context. When the use background flag is `GR_TRUE` the background color is used when drawing text or bitmaps. When the flag is `GR_FALSE` the background color is not used when drawing text and bitmaps.

Parameters

Type	Name	Description
GR_GC_ID	*gc*	The ID of the graphics context to modify.
GR_BOOL	*flag*	The new value for the GC's use background flag.

See Also

`GrNewGC()`, `GrGetGCInfo()`, `GrSetGCBackground()`.

GrSetScreenSaverTimeout()

Name

`GrSetScreenSaverTimeout()` — Set screen saver timeout

Synopsis

```
void  GrSetScreenSaverTimeout ( GR_TIMEOUT  timeout );
```

Description

This function sets the timeout period of the system's screen saver event.

> **Note:** A bug in version 0.89-pre7 of Microwindows causes the timeout to be multiplied internally by 1000. Therefore when building with 0.89-pre7 you must set the timeout in seconds rather than milli-seconds.

Parameters

Type	Name	Description
GR_TIMEOUT	*timeout*	The screen saver timeout period in milliseconds. If no pointer or keyboard input occurs for the duration of this timeout period, a `GR_EVENT_SCREENSAVER` event is sent to each window that has selected the event.

Example

The following example will turn an LCD backlight off when a 60 second screen saver timer expires, and restore the backlight when user input is resumed.

Example 2-1. Using `GrSetScreenSaverTimeout()`

```
void setup_screensaver (void)
{
    /* Set a one minute timeout for the LCD backlight */
    /* nano-X BUG: use seconds rather than mS for 0.89-pre7 */
    GrSetScreenSaverTimeout (60 * 1000);
}

void process_screensaver_event (GR_EVENT_SCREENSAVER *event)
{
    if (event->activate)
    {
        /* Turn the LCD backlight off */
        your_platforms_backlight_off();
    }
    else
    {
        /* Turn the LCD backlight on */
        your_platforms_backlight_on();
    }
}
```

See Also

GR_EVENT_SCREENSAVER.

GrSetSystemPalette()

Name

`GrSetSystemPalette()` — Set the colors of the system palette

Synopsis

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

Description

This function copies the colors in the specified palette into the system palette. All colors in *`pal`* are copied to the system palette. The first palette entry in *`pal`* is copied into the system palette at the index specified by *`first`*. Therefore all existing entries in the system palette before *`first`* will remain unchanged.

For example if the system palete has 50 colors defined, and you use `GrSetSystemPalette()` to add a 50 color palette. If you specify *`first`* as 50, then the resulting system palette will have 100 colors. If you specify *`first`* as 20, then you will have a 70 color palette, and the last 30 colors of the original system palette will be overwritten with new colors.

Parameters

Type	Name	Description
GR_COUNT	*`first`*	The first palette entry in the system palette to receive new colors from the new palette.
GR_PALETTE	*`pal`*	The new color palette.

See Also

`GrGetSystemPalette()`, `GrFindColor()`, `GrGetSysColor()`.

`GrSetWMProperties()`

Name

`GrSetWMProperties()` — Set a window's properties

Synopsis

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

Description

This function sets the specified window's window manager properties to the properties specified in *`props`*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*`wid`*	The ID of the window to set the properties of.

Type	Name	Description
GR_WM_PROPERTIES*	*props*	A pointer to the caller supplied structure which contains the new window manager properties.

See Also

`GrGetWMProperties()`, `GrNewWindowEx()`, `GrSetWindowBackgroundColor()`, `GrSetWindowBorderSize()`, `GrSetWindowBorderColor()`, `GrSetWindowTitle()`.

GrSetWindowBackgroundColor()

Name

`GrSetWindowBackgroundColor()` — Set a window's background color

Synopsis

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

Description

This macro sets the specified window's background color to *`color`*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to set the background color of.
GR_COLOR	*color*	The color to set the window background to.

See Also

`GrSetWMProperties()`, `GrGetWMProperties()`, `GR_RGB()`.

GrSetWindowBorderColor()

Name

`GrSetWindowBorderColor()` — Set a window's border color

Synopsis

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

Description

This macro sets the specified window's border color to *`color`*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*`wid`*	The ID of the window to set the border color of.
GR_COLOR	*`color`*	The color to set the window border to.

See Also

`GrSetBorderColor()`, `GrSetWMProperties()`, `GrGetWMProperties()`, `GR_RGB()`.

`GrSetWindowBorderSize()`

Name

`GrSetWindowBorderSize()` — Set a window's border width

Synopsis

```
void  GrSetWindowBorderSize ( GR_WINDOW_ID  wid ,  GR_SIZE
```

```
width );
```

Description

This macro sets the specified window's border width to *width*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to set the border width of.
GR_SIZE	*width*	The width, in pixels, to set the window border to.

See Also

`GrSetWMProperties()`, `GrGetWMProperties()`.

GrSetWindowTitle()

Name

`GrSetWindowTitle()` — Set a window's title

Synopsis

```
void  GrSetWindowTitle ( GR_WINDOW_ID  wid ,  GR_CHAR *  name );
```

Description

This macro sets the specified window's title to *name*.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to set the border width of.
GR_CHAR*	*name*	The text string to set the window title to.

See Also

`GrSetWMProperties()`, `GrGetWMProperties()`.

GrSubtractRegion()

Name

`GrSubtractRegion()` — Form a region from the difference of two regions

Synopsis

```
void  GrSubtractRegion ( GR_REGION_ID  dst_rgn ,  GR_REGION_ID  src_rgn1 ,  GR_REGION_ID  src_rgn2 );
```

Description

This function creates a region from the two specified source regions and places the resulting region in the destination region. The resulting region is *dst_rgn* = *src_rgn1* - *src_rgn2*.

Parameters

Type	Name	Description
GR_REGION_ID	*dst_rgn*	The ID of the destination region.
GR_REGION_ID	*src_rgn1*	The ID of the the first of two source regions.
GR_REGION_ID	*src_rgn2*	The ID of the second of two source regions.

See Also

GrUnionRectWithRegion(), GrUnionRegion(), GrXorRegion(), GrIntersectRegion(), GrDestroyRegion(),

GrText()

Name

GrText() — Draw text

Synopsis

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

Description

This function draws the text *str* at position (*x*, *y*) on the specified drawable. The text will be drawn in the foreground color of the graphics context *gc*. If the *usebackground* flag of the GC is set, then the text background will be drawn in the GC's background color. If the *usebackground* flag is clear, then the text background will not be drawn.

Parameters

Type	Name	Description
GR_DRAW_ID	*id*	The ID of the drawable to draw the text onto.
GR_GC_ID	*gc*	The ID of the graphics context to use when drawing the text.

Type	Name	Description
GR_COORD	*x*	The X coordinate to draw the text at, relative to the drawable.
GR_COORD	*y*	The Y coordinate to draw the text at, relative to the drawable.
void*	*str*	The input string. If the string is *NOT zero terminated, then the length of the string must be specified in the* `count parameter.`
GR_COUNT	*count*	The length of the string. This parameter can be set to `-1 if the string is zero terminated and` *flags contains GR_TFASCII.*
int	*fags*	Text rendering flags, can be a combination of flags listed below.

The flags parameter is a combination of flags from the following three groups. The combination can include one string encoding flag, one alignment flag and multiple attribute flags.

String encoding flags:

GR_TFASCII	GR_TFUTF8	GR_TFUC16	GR_TFUC32

Text alignment flags:

GR_TFTOP	GR_TFBASELINE	GR_TFBOTTOM

Text attribute flags:

GR_TFKERNING	GR_TFANTIALIAS	GR_TFUNDERLINE

Example

Example 2-1. Using `GrText()`

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
```

```
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

See Also

`GrSetGCForeground()`, `GrSetGCBackground()`, `GrSetGCUseBackground()`, `GrGetGCTextSize()`.

`GrUnionRectWithRegion()`

Name

`GrUnionRectWithRegion()` — Form union of rectangle and region

Synopsis

```
void  GrUnionRectWithRegion ( GR_REGION_ID  region ,  GR_RECT *
rect );
```

Description

This function forms a union of the specified region and the region defined by the specified rectangle. The resulting area is placed back into the original source region.

Parameters

Type	Name	Description
GR_REGION_ID	*region*	The ID of the region to modify.
GR_RECT*	*rect*	A pointer to the rectangle to merge into the region.

See Also

`GrUnionRegion()`, `GrGetRegionBox()`.

GrUnionRegion()

Name

`GrUnionRegion()` — Form a region from the union of two other regions

Synopsis

```
void  GrUnionRegion ( GR_REGION_ID  dst_rgn ,  GR_REGION_ID  src_rgn1 ,  GR_REGION_ID  src_rgn2 );
```

Description

This function creates a union of the two specified source regions and places the resulting region in the destination region.

Parameters

Type	Name	Description
GR_REGION_ID	*dst_rgn*	The ID of the destination region.
GR_REGION_ID	*src_rgn1*	The ID of the the first of two source regions.
GR_REGION_ID	*src_rgn2*	The ID of the second of two source regions.

See Also

`GrUnionRectWithRegion()`, `GrSubtractRegion()`, `GrXorRegion()`, `GrIntersectRegion()`, `GrDestroyRegion()`,

GrUnmapWindow()

Name

GrUnmapWindow() — Unmap a window and it's children

Synopsis

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

Description

This function recursively unmaps (hides) the specified window and all of its child windows.

Parameters

Type	Name	Description
GR_WINDOW_ID	*wid*	The ID of the window to unmap.

See Also

GrNewWindow(), GrMapWindow(), GrGetWindowInfo().

GrXorRegion()

Name

GrXorRegion() — Form a region form the XOR two regions

Synopsis

```
void  GrXorRegion ( GR_REGION_ID  dst_rgn ,  GR_REGION_ID
src_rgn1 ,  GR_REGION_ID  src_rgn2 );
```

Description

This function creates a region from the two specified source regions and places the resulting region in the destination region. The resulting region is *dst_rgn* = *src_rgn1* XOR *src_rgn2*.

Parameters

Type	Name	Description
GR_REGION_ID	*dst_rgn*	The ID of the destination region.
GR_REGION_ID	*src_rgn1*	The ID of the the first of two source regions.
GR_REGION_ID	*src_rgn2*	The ID of the second of two source regions.

See Also

`GrUnionRegion()`, `GrSubtractRegion()`, `GrIntersectRegion()`, `GrDestroyRegion()`,

Chapter 3. Nano-X Data Types

GR_BITMAP

Name

GR_BITMAP — Bitmap unit

Synopsis

```
typedef unsigned short  GR_BITMAP;
```

Description

The GR_BITMAP type is used to specify small monochrome bitmapped images. These bitmap images are generally used for mouse cursors.

Each pixel is represented by a bit in a GR_BITMAP value. The bitmaps can be up to 16x16 pixels in size. Therefore this type will define a complete row of the bitmap within one GR_BITMAP value. To define a 16x16 bitmap, you would use a 16 element array of GR_BITMAP values.

Example

The following example builds an X shaped mouse cursor.

Example 3-1. Using GR_BITMAP

```
void set_x_cursor (GR_WINDOW_ID wid)
{
    GR_BITMAP fg_bitmap[16];
    GR_BITMAP bg_bitmap[16];

    fg_bitmap[0]  = 0x8001;  /* X______________X */
    fg_bitmap[1]  = 0x4002;  /* _X____________X_ */
    fg_bitmap[2]  = 0x2004;  /* __X__________X__ */
    fg_bitmap[3]  = 0x1008;  /* ___X________X___ */
    fg_bitmap[4]  = 0x0810;  /* ____X______X____ */
    fg_bitmap[5]  = 0x0420;  /* _____X____X_____ */
    fg_bitmap[6]  = 0x0240;  /* ______X__X______ */
    fg_bitmap[7]  = 0x0180;  /* _______XX_______ */
    fg_bitmap[8]  = 0x0180;  /* _______XX_______ */
    fg_bitmap[9]  = 0x0240;  /* ______X__X______ */
    fg_bitmap[10] = 0x0420;  /* _____X____X_____ */
    fg_bitmap[11] = 0x0810;  /* ____X______X____ */
    fg_bitmap[12] = 0x1008;  /* ___X________X___ */
    fg_bitmap[13] = 0x2004;  /* __X__________X__ */
    fg_bitmap[14] = 0x4002;  /* _X____________X_ */
    fg_bitmap[15] = 0x8001;  /* X______________X */

    bg_bitmap[0]  = 0x4002;  /* _X____________X_ */
    bg_bitmap[1]  = 0xA005;  /* X_X__________X_X */
    bg_bitmap[2]  = 0x500A;  /* _X_X________X_X_ */
    bg_bitmap[3]  = 0x2814;  /* __X_X______X_X__ */
    bg_bitmap[4]  = 0x1428;  /* ___X_X____X_X___ */
    bg_bitmap[5]  = 0x0A50;  /* ____X_X__X_X____ */
    bg_bitmap[6]  = 0x05A0;  /* _____X_XX_X_____ */
    bg_bitmap[7]  = 0x0240;  /* ______X__X______ */
    bg_bitmap[8]  = 0x0240;  /* ______X__X______ */
    bg_bitmap[9]  = 0x05A0;  /* _____X_XX_X_____ */
    bg_bitmap[10] = 0x0A50;  /* ____X_X__X_X____ */
    bg_bitmap[11] = 0x1428;  /* ___X_X____X_X___ */
    bg_bitmap[12] = 0x2814;  /* __X_X______X_X__ */
    bg_bitmap[13] = 0x500A;  /* _X_X________X_X_ */
```

```
    bg_bitmap[14] = 0xA005;  /* X_X__________X_X */
    bg_bitmap[15] = 0x4002;  /* _X____________X_ */

    GrSetCursor (wid, 16, 16, 8, 8,
                 BLACK, BLACK, fg_bitmap, bg_bitmap);
}
```

See Also

`GrSetCursor()`, `GrBitmap()`.

GR_BOOL

Name

`GR_BOOL` — Boolean type

Synopsis

```
typedef unsigned short  GR_BOOL;
```

Description

The GR_BOOL type represents a boolean value within nano-X. It can be one of `GR_TRUE` or `GR_FALSE`.

GR_BUTTON

Name

GR_BUTTON — Mouse button codes

Synopsis

```
typedef unsigned int GR_BUTTON;
```

Description

A GR_BUTTON type is a bitwise OR combination of one or more of the following flags. The set flags indicate the mouse buttons that are pressed.

Table 3-1. Mouse Buton Enumerations

Value	Description
GR_BUTTON_R	Indicates the right mouse button.
GR_BUTTON_M	Indicates the middle mouse button.
GR_BUTTON_L	Indicates the left mouse button.
GR_BUTTON_ANY	Bitwise OR of all the valid mouse button flags.

GR_CHAR

Name

GR_CHAR — Text character

Synopsis

```
typedef unsigned char  GR_CHAR;
```

Description

The GR_CHAR type is used for ASCII text, filenames and keystrokes.

GR_CHAR_WIDTH

Name

GR_CHAR_WIDTH — Character width

Synopsis

```
typedef unsigned char  GR_CHAR_WIDTH;
```

Description

The GR_CHAR_WIDTH type describes the width of a character in pixels.

GR_COORD

Name

GR_COORD — Coordinate value

Synopsis

```
typedef int  GR_COORD;
```

Description

The GR_COORD type is typically used to specify the X or Y location of a graphic object relative to its parent window or relative to the screen.

GR_COLOR

Name

`GR_COLOR` — Color value

Synopsis

```
typedef unsigned long GR_COLOR;
```

Description

A GR_COLOR type is a device independent value used to define a color within nano-X.

Nano-X uses an unsigned 32 bit integer to represent colors. The lowest order three bytes define the color, while the highest order byte is always `0`. This results in approximately 16 million colors that can be sopecified. The colorvalue is laid out as:

Table 3-1. 32bit Color Value

31 . . . 24	23 . . . 16	15 . . . 8	7 . . . 0
0	Blue	Green	Red

The macro `GR_RGB()` can be used to specify a color.

Color Definitions

If you define `MWINCLUDECOLORS` before inclusion of the nano-X header file.

```
#define MWINCLUDECOLORS
#include "nano-X.h"
```

Your application will have access to the following basic color definitions.

Table 3-2. Basic Color Definitions

BLACK	BLUE	GREEN	CYAN	RED
MAGENTA	BROWN	LTGRAY	LTBLUE	LTGREEN
LTCYAN	LTRED	LTMAGENTA	YELLOW	WHITE
DKGRAY				

The following definitions are not actually colors of GR_COLOR but they can be used with the `GrGetSysColor()` to retrieve a GR_COLOR type value.

Table 3-3. System Color Definitions

Color Index	Description
GR_COLOR_DESKTOP	Desktop background color
GR_COLOR_ACTIVECAPTION	Active window caption color
GR_COLOR_ACTIVECAPTIONTEXT	Active window caption text color
GR_COLOR_INACTIVECAPTION	Inactive window caption color
GR_COLOR_INACTIVECAPTIONTEXT	Inactive window caption text color
GR_COLOR_WINDOWFRAME	3-D Window frame color
GR_COLOR_BTNSHADOW	3-D button shadow color

Color Index	Description
GR_COLOR_3DLIGHT	3-D window light color
GR_COLOR_BTNHIGHLIGHT	3-D button highlight color
GR_COLOR_APPWINDOW	Top level window background color
GR_COLOR_APPTEXT	Top level window text color
GR_COLOR_BTNFACE	Button face color
GR_COLOR_BTNTEXT	Button text color
GR_COLOR_WINDOW	Normal backgrouns color in a window
GR_COLOR_WINDOWTEXT	Normal text color in a window.
GR_COLOR_HIGHLIGHT	Highlight background color
GR_COLOR_HIGHLIGHTTEXT	Highlighted text color
GR_COLOR_GRAYTEXT	Grayed out text color
GR_COLOR_MENUTEXT	Menu text color
GR_COLOR_MENU	Menu background color

See Also

`GR_RGB()`, `GrGetSysColor()`, `GrSetBorderColor()`, `GrSetGCForeground()`, `GrSetGCBackground()`, `GrFindColor()`, `GrSetWindowBackgroundColor()`, `GrSetWindowBorderColor()`.

GR_COUNT

Name

GR_COUNT — Number of items

Synopsis

```
typedef int  GR_COUNT;
```

Description

The GR_COUNT type is typically used to specify the number of elements in an array. The polygon functions use this type to specify the number of points in the array of vertices. Some of the functions with string parameters use GR_COUNT to specify the number of characters in non-zero terminated strings.

GR_DRAW_ID

Name

GR_DRAW_ID — Drawable ID

Synopsis

```
typedef GR_ID  GR_DRAW_ID;
```

Description

The GR_DRAW_ID type uniquely identifies a nano-X drawable object. Windows and pixmaps are drawable objects.

See Also

GR_ID, GR_WINDOW_ID.

GR_ERROR

Name

GR_ERROR — Error event codes

Synopsis

```
typedef int GR_ERROR;
```

Description

A GR_ERROR enumeration type identifies the cause of an error event. When a window receives an error event, the associated GR_EVENT_ERROR structure contains a field of this type that specifies the error.

The following table shows all of the available values that may be assigned to a GR_ERROR variable.

Table 3-1. Error Codes

Value	Description
GR_ERROR_BAD_WINDOW_ID	A function call was made into the nano-X library with an invalid window ID specified.
GR_ERROR_BAD_GC_ID	A function call was made into the nano-X library with an invalid graphics context ID specified.
GR_ERROR_BAD_CURSOR_SIZE	A cursor with an invalid size was specified.
GR_ERROR_MALLOC_FAILED	A memory allocation within the server failed.
GR_ERROR_BAD_WINDOW_SIZE	An invalid window size was specified.
GR_ERROR_KEYBOARD_ERROR	An error occred while the server was reading from the keyboard.
GR_ERROR_MOUSE_ERROR	An error occred while the server was reading from the mouse.
GR_ERROR_INPUT_ONLY_WINDOW	A graphics drawing function was illegaly invoked on an input only window.
GR_ERROR_ILLEGAL_ON_ROOT_WINDOW	An illegal attempt was made to perform an operation that can not be performed on the "root" window.

Value	Description
GR_ERROR_TOO_MUCH_CLIPPING	The clipping region became too complex for nano-X to handle.
GR_ERROR_SCREEN_ERROR	An error occred while the server was writing to the screen driver.
GR_ERROR_UNMAPPED_FOCUS_WINDOW	An illegal attempt was made to set focus to an unmapped window.
GR_ERROR_BAD_DRAWING_MODE	An invalid drawing mode was specified to a graphics context.

See Also

GR_EVENT_ERROR, GrDefaultErrorHandler().

GR_EVENT

Name

GR_EVENT — Generic event structure

Synopsis

```
typedef union
{
    GR_EVENT_TYPE               type;
    GR_EVENT_ERROR              error;
    GR_EVENT_GENERAL            general;
```

```
    GR_EVENT_BUTTON            button;
    GR_EVENT_KEYSTROKE         keystroke;
    GR_EVENT_EXPOSURE          exposure;
    GR_EVENT_MOUSE             mouse;
    GR_EVENT_FDINPUT           fdinput;
    GR_EVENT_UPDATE            update;
    GR_EVENT_SCREENSAVER       screensaver;
    GR_EVENT_CLIENT_DATA_REQ   clientdatareq;
    GR_EVENT_CLIENT_DATA       clientdata;
    GR_EVENT_SELECTION_CHANGED selectionchanged;
} GR_EVENT;
```

Description

The GR_EVENT structure is used to retrieve event information from the nano-X event queue. When you pull an event out of the event queue you don't know what type of event it is until after you have the event. Various event structures are different sizes, this structure is a union of all event types. Since this structure is a union, it is guarenteed to be large enough to hold the largest possible event when you get an event from the event queue.

The *`type`* field identifies the structure type. After receiving an event it is common for an application to switch on type.

Fields

Type	Name	Description
GR_EVENT_TYPE	type	The type of event that this structure corresponds too.

Type	Name	Description
GR_EVENT_ERROR	error	Additional event data, if the event type is GR_EVENT_TYPE_ERROR.
GR_EVENT_GENERAL	general	Additional event data, if the event type is GR_EVENT_TYPE_CLOSE_REQ, GR_EVENT_TYPE_MOUSE_EXIT, GR_EVENT_TYPE_MOUSE_ENTER, GR_EVENT_TYPE_FOCUS_OUT or GR_EVENT_TYPE_FOCUS_IN.
GR_EVENT_BUTTON	button	Additional event data, if the event type is GR_EVENT_TYPE_BUTTON_UP or GR_EVENT_TYPE_BUTTON_DOWN.
GR_EVENT_KEYSTROKE	keystroke	Additional event data, if the event type is GR_EVENT_TYPE_KEY_DOWN or GR_EVENT_TYPE_KEY_UP.
GR_EVENT_EXPOSURE	exposure	Additional event data, if the event type is GR_EVENT_TYPE_EXPOSURE.
GR_EVENT_MOUSE	mouse	Additional event data, if the event type is GR_EVENT_TYPE_MOUSE_ENTER, GR_EVENT_TYPE_MOUSE_EXIT, GR_EVENT_TYPE_MOUSE_MOTION or GR_EVENT_TYPE_Mouse_POSITION.
GR_EVENT_FDINPUT	fdinput	Additional event data, if the event type is GR_EVENT_TYPE_FDINPUT.
GR_EVENT_UPDATE	update	Additional event data, if the event type is GR_EVENT_TYPE_UPDATE.

Type	Name	Description
GR_EVENT_SCREENSAVER	screensaver	Additional event data, if the event type is `GR_EVENT_TYPE_SCREENSAVER`.
GR_EVENT_CLIENT_DATA_REQ	clientdatareq	Additional event data, if the event type is `GR_EVENT_TYPE_CLIENT_DATA_REQ`.
GR_EVENT_CLIENT_DATA	clientdata	Additional event data, if the event type is `GR_EVENT_TYPE_CLIENT_DATA`.
GR_EVENT_SELECTION_CHANGED	selectionchanged	Additional event data, if the event type is `GR_EVENT_TYPE_SELECTION_CHANGED`.

Example

The following example shows a typical event loop. The first line of the infinite while loop will suspend the client application until an event is available in the event queue. Then the example switches on the event type calling the appropriate application function to process the event.

Example 3-1. Using GR_EVENT

```
void typical_event_loop (void)
{
    GR_EVENT  event;

    while (1)
    {
        GrGetNextEvent (&event);
        switch (event.type)
        {
        case GR_EVENT_TYPE_EXPOSURE:
```

```
            process_exposure_event ((GR_EVENT_EXPOSURE*) event);
            break;

        case GR_EVENT_TYPE_BUTTON_DOWN:
            process_button_event ((GR_EVENT_BUTTON*) event);
            break;

        case GR_EVENT_TYPE_KEY_DOWN:
        case GR_EVENT_TYPE_KEY_UP:
            process_key_event ((GR_EVENT_KEYSTROKE*) event);
            break;

        case GR_EVENT_TYPE_SCREENSAVER:
            pro-
cess_screensaver_event ((GR_EVENT_SCREENSAVER*) event);
            break;

        case GR_EVENT_TYPE_CLOSE_REQ:
            GrClose();
            exit (0);
        }
    }
}
```

See Also

`GrMainLoop()`, `GrGetNextEvent()`, `GrGetNextEventTimeout()`, `GrCheckNextEvent()`, `GrPeekEvent()`.

GR_EVENT_BUTTON

Name

GR_EVENT_BUTTON — Mouse button event structure

Synopsis

```
typedef struct
{
    GR_EVENT_TYPE   type;
    GR_WINDOW_ID    wid;
    GR_WINDOW_ID    subwid;
    GR_COORD        rootx;
    GR_COORD        rooty;
    GR_COORD        x;
    GR_COORD        y;
    GR_BUTTON       buttons;
    GR_BUTTON       changebuttons;
    GR_KEYMOD       modifiers;
    GR_TIMEOUT      time;
} GR_EVENT_BUTTON;
```

Description

The GR_EVENT_BUTTON structure is used by nano-X to report changes in the status of the mouse buttons. When a mouse button state changes only one mouse button event is sent to a client. The event is sent to the highest window that has selected for the event. If the window's parent has also selected for button events, nano-X will not send an additional event for the parent window.

If a window has selected both `GR_EVENT_TYPE_BUTTON_DOWN` and `GR_EVENT_TYPE_BUTTON_UP` events, nano-X will grab the mouse for that window when a mouse button is first pressed down. While the mouse is grabbed, no mouse button or position events will be delivered to any window besides the window that nano-X grabbed the mouse for. The mouse will remain grabbed until all of the mouse buttons are released.

Fields

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be either a `GR_EVENT_TYPE_BUTTON_DOWN` or a `GR_EVENT_TYPE_BUTTON_UP` type.
GR_WINDOW_ID	wid	The ID of the window that the mouse button event is being sent to. If the mouse has been grabbed then this is the window that nano-X grabbed the mouse for. In this case the mouse may not actually be positioned over the window any longer. The mouse may be over a child window or it may be outside the window that grabbed the mouse.
GR_WINDOW_ID	subwid	The ID of the window that the mouse button event occurs in. Generally this field will be the same as *`wid, but in some cases if the mouse event occurs in a decendant of wid, then this field indicates that child window.`*
GR_COORD	rootx	The X coordinate of the mouse pointer relative to the root window.

Type	Name	Description
GR_COORD	rooty	The Y coordinate of the mouse pointer relative to the root window.
GR_COORD	x	The X coordinate of the mouse pointer relative to the window *wid.*
GR_COORD	y	The Y coordinate of the mouse pointer relative to the window *wid.*
GR_BUTTON	buttons	Indicates the buttons that are being pressed.
GR_BUTTON	changebuttons	Indicates the buttons that have just changed state. If the event type is GR_EVENT_TYPE_BUTTON_DOWN, then this field indicates the button(s) that were just pressed. If the event type is GR_EVENT_TYPE_BUTTON_UP, then this field indicates the button(s) that were just released.
GR_KEYMOD	modifiers	Indicates the status of the keyboard modifier keys.
GR_TIMEOUT	time	Time stamp of when the button event occured in milliseconds.

See Also

GR_EVENT, GR_EVENT_MOUSE, GR_EVENT_KEYSTROKE.

GR_EVENT_ERROR

Name

GR_EVENT_ERROR — Error event structure

Synopsis

```
typedef struct
{
    GR_EVENT_TYPE   type;
    GR_FUNC_NAME    name;
    GR_ERROR        code;
    GR_ID           id;
} GR_EVENT_ERROR;
```

Description

The GR_EVENT_ERROR structure is used by nano-X to report runtime errors. Some errors are system errors, such as GR_ERROR_MALLOC_FAILED which indicates that a memory allocation failed. Other error types are program errors, such as GR_ERROR_ILLEGAL_ON_ROOT_WINDOW which will occur if for example the program tries to move the root window.

Fields

Type	Name	Description

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be `GR_EVENT_TYPE_ERROR`.
GR_FUNC_NAME	name	The name of the function in which the error occured.
GR_ERROR	code	The type of error that occured.
GR_ID	id	The window ID of the window that an error occured on, if the event is related to a window. The GC ID of the graphics context that an error occured on if the error occured on a GC. Set to `0 if the error is not related to a window or a GC.`

Example

The following example shows a typical error handler.

Example 3-1. Using GR_EVENT_ERROR

```
char *error_strings[] = { GR_ERROR_STRINGS };  /* See nano-
X.h */

void  process_error_event (GR_EVENT_ERROR *event)
{
    printf ("NANO-X ERROR (function %s): ", event->name);
    printf (error_strings[event->code], event->id);
    printf ("\n");
    fflush (stdout);

    GrClose();
    exit (1);
}
```

See Also

GR_EVENT, `GrDefaultErrorHandler()`.

GR_EVENT_EXPOSURE

Name

`GR_EVENT_EXPOSURE` — Window exposure event structure

Synopsis

```
typedef struct
{
    GR_EVENT_TYPE   type;
    GR_WINDOW_ID    wid;
    GR_COORD        x;
    GR_COORD        y;
    GR_SIZE         width;
    GR_SIZE         height;
} GR_EVENT_EXPOSURE;
```

Description

The GR_EVENT_EXPOSURE structure is used by nano-X to tell the application that a portion of a window has just become visible and must be redrawn. This event is sent immediately after a window is mapped or after an obstructing window is moved.

The event structure contains an exposure rectangle. Only the contents of this rectangle need to be redrawn. It does not generally hurt to redraw the entire window, but a performance boost may be achieved by limiting the amount of redrawing that the application performs.

Fields

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be `GR_EVENT_TYPE_EXPOSURE`.
GR_WINDOW_ID	wid	The ID of the window that is being exposed.
GR_COORD	x	The X coordinate of upper left corner of the exposed rectangle relative to the upper left corner of the window specified by *`wid`*.
GR_COORD	y	The Y coordinate of upper left corner of the exposed rectangle relative to the upper left corner of the window specified by *`wid`*.
GR_SIZE	width	The width of the exposed rectangle.
GR_SIZE	height	The height of the exposed rectangle.

Example

Example 3-1. Using GR_EVENT_TYPE_EXPOSURE

```
#include <stdio.h>
#define MWINCLUDECOLORS
```

```
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
```

```
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

See Also

GR_EVENT.

GR_EVENT_FDINPUT

Name

GR_EVENT_FDINPUT — File descriptor input event structure

Synopsis

```
typedef struct
{
    GR_EVENT_TYPE   type;
    int             fd;
} GR_EVENT_FDINPUT;
```

Description

The GR_EVENT_FDINPUT structure is used by nano-X to tell the application that data is available for reading on a file descriptor that the application previously registered with the function `GrRegisterInput()`.

Fields

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be `GR_EVENT_TYPE_FDINPUT`.
int	fd	The file descriptor that has data available for reading.

See Also

GR_EVENT, `GrRegisterInput()`,

GR_EVENT_GENERAL

Name

`GR_EVENT_GENERAL` — General purpose event structure

Synopsis

```
typedef struct
{
    GR_EVENT_TYPE   type;
    GR_WINDOW_ID    wid;
    GR_WINDOW_ID    otherid;
} GR_EVENT_GENERAL;
```

Description

The GR_EVENT_GENERAL structure is used by nano-X to pass data related to the events:

- `GR_EVENT_TYPE_TIMEOUT`
- `GR_EVENT_TYPE_CLOSE_REQ`
- `GR_EVENT_TYPE_MOUSE_EXIT`
- `GR_EVENT_TYPE_MOUSE_ENTER`
- `GR_EVENT_TYPE_FOCUS_OUT`
- `GR_EVENT_TYPE_FOCUS_IN`

Each of these events use the *`type`* and *`wid`* fields. The third field (*`otherid`* is only used for the focus events.

Fields

Type	Name	Description

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be one of `GR_EVENT_TYPE_TIMEOUT`, `GR_EVENT_TYPE_CLOSE_REQ`, `GR_EVENT_TYPE_MOUSE_ENTER`, `GR_EVENT_TYPE_MOUSE_EXIT`, `GR_EVENT_TYPE_FOCUS_IN`, `GR_EVENT_TYPE_FOCUS_OUT`.
GR_WINDOW_ID	wid	For `GR_EVENT_TYPE_CLOSE_REQ` events this is the ID of the window that is being closed. For `GR_EVENT_TYPE_MOUSE_ENTER` and `GR_EVENT_TYPE_MOUSE_EXIT` events this is the ID of the window the mouse is entering or exiting, respectively. For `GR_EVENT_TYPE_FOCUS_IN` events this is the ID of the window that is receiving the focus. For `GR_EVENT_TYPE_FOCUS_OUT` events this is the window that is loosing the focus. This field is unused for `GR_EVENT_TYPE_TIMEOUT` events.

Type	Name	Description
GR_ID	otherid	For GR_EVENT_TYPE_FOCUS_IN events this is the ID of the window that is loosing the focus. For GR_EVENT_TYPE_FOCUS_OUT events this is the ID of the window that is receiving the focus. This field is unused for all other event types.

Example

Example 3-1. Using GR_EVENT_TYPE_CLOSE_REQ

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
```

```
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

See Also

GR_EVENT.

GR_EVENT_KEYSTROKE

Name

GR_EVENT_KEYSTROKE — Keyboard event structure

Synopsis

```
typedef struct
{
    GR_EVENT_TYPE   type;
    GR_WINDOW_ID    wid;
    GR_WINDOW_ID    subwid;
    GR_COORD        rootx;
    GR_COORD        rooty;
    GR_COORD        x;
    GR_COORD        y;
    GR_BUTTON       buttons;
    GR_KEYMOD       modifiers;
    GR_KEY          ch;
    GR_SCANCODE     scancode;
} GR_EVENT_KEYSTROKE;
```

Description

The GR_EVENT_KEYSTROKE structure is used by nano-X to pass the application keyboard events.

The keystroke will be sent to the highest window that contains the mouse cursor and has selected to receive keystroke events, if that window is a decendant of the focus

window. Otherwise the keystroke is sent to the focus window or it's highest ancestor that has selected to receive keystroke events.

Fields

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be either a `GR_EVENT_TYPE_KEY_DOWN` or a `GR_EVENT_TYPE_KEY_UP` type.
GR_WINDOW_ID	wid	The ID of the window that the keystroke event is being sent to.
GR_WINDOW_ID	subwid	The ID of the window that the mouse is in. Generally this field will be the same as *`wid, but in some cases if the mouse event occurs in a decendant of wid, then this field indicates that child window.`*
GR_COORD	rootx	The X coordinate of the mouse pointer relative to the root window.
GR_COORD	rooty	The Y coordinate of the mouse pointer relative to the root window.
GR_COORD	x	The X coordinate of the mouse pointer relative to the window *`wid.`*
GR_COORD	y	The Y coordinate of the mouse pointer relative to the window *`wid.`*
GR_BUTTON	buttons	Indicates the mouse buttons that are being pressed.
GR_KEYMOD	modifiers	Indicates the status of the keyboard modifier keys.

Type	Name	Description
GR_KEY	ch	The key that caused the keystroke event.
GR_SCANCODE	scancode	The OEM scancode for the key if it is available.

Example

The following example will print all keystroke codes to the console.

Example 3-1. Using GR_EVENT_KEYSTROKE

```
void process_key_event (GR_EVENT_KEYSTROKE *event)
{
    printf ("%s  MOD:0x%08X  CH:0x%04X  SCAN:0x%02X\n",
            (event-
>type == GR_EVENT_TYPE_KEY_DOWN) ? "DN" : "UP",
            event->modifiers, event->ch, event->scancode);
    fflush (stdout);
}
```

See Also

GR_EVENT, GR_EVENT_MOUSE.

GR_EVENT_MASK

Name

GR_EVENT_MASK — Event masks

Synopsis

```
typedef unsigned long GR_EVENT_MASK;
```

Description

A GR_EVENT_MASK type is a bitwise OR combination of one or more of the following event mask flags. This type is used along with the `GrSelectEvents()` to select which event types a window will receive.

Table 3-1. Event Mask Bits

Value	Description
GR_EVENT_MASK_NONE	This flag consists on NO event flags.
GR_EVENT_MASK_ALL	This mask is a combination of all other mask flags.
GR_EVENT_MASK_ERROR	When set the window may receive GR_EVENT_TYPE_ERROR events.
GR_EVENT_MASK_EXPOSURE	When set the window may receive GR_EVENT_TYPE_EXPOSURE events.
GR_EVENT_MASK_BUTTON_DOWN	When set the window may receive GR_EVENT_TYPE_BUTTON_DOWN events.

Value	Description
GR_EVENT_MASK_BUTTON_UP	When set the window may receive GR_EVENT_TYPE_BUTTON_UP events.
GR_EVENT_MASK_MOUSE_ENTER	When set the window may receive GR_EVENT_TYPE_MOUSE_ENTER events.
GR_EVENT_MASK_MOUSE_EXIT	When set the window may receive GR_EVENT_TYPE_MOUSE_EXIT events.
GR_EVENT_MASK_MOUSE_MOTION	When set the window may receive GR_EVENT_TYPE_MOUSE_MOTION events.
GR_EVENT_MASK_MOUSE_POSITION	When set the window may receive GR_EVENT_TYPE_MOUSE_POSITION events.
GR_EVENT_MASK_KEY_DOWN	When set the window may receive GR_EVENT_TYPE_KEY_DOWN events.
GR_EVENT_MASK_KEY_UP	When set the window may receive GR_EVENT_TYPE_KEY_UP events.
GR_EVENT_MASK_FOCUS_IN	When set the window may receive GR_EVENT_TYPE_FOCUS_IN events.
GR_EVENT_MASK_FOCUS_OUT	When set the window may receive GR_EVENT_TYPE_FOCUS_OUT events.
GR_EVENT_MASK_FDINPUT	When set the window may receive GR_EVENT_TYPE_FDINPUT events.
GR_EVENT_MASK_UPDATE	When set the window may receive GR_EVENT_TYPE_UPDATE events.
GR_EVENT_MASK_CHLD_UPDATE	When set the window may receive GR_EVENT_TYPE_CHLD_UPDATE events.
GR_EVENT_MASK_CLOSE_REQ	When set the window may receive GR_EVENT_TYPE_CLOSE_REQ events.
GR_EVENT_MASK_TIMEOUT	When set the window may receive GR_EVENT_TYPE_TIMEOUT events.

Value	Description
GR_EVENT_MASK_SCREENSAVER	When set the window may receive GR_EVENT_TYPE_SCREENSAVER events.
GR_EVENT_MASK_CLIENT_DATA_REQ	When set the window may receive GR_EVENT_TYPE_CLIENT_DATA_REQ events.
GR_EVENT_MASK_CLIENT_DATA	When set the window may receive GR_EVENT_TYPE_CLIENT_DATA events.
GR_EVENT_MASK_SELECTION_CHANGED	When set the window may receive GR_EVENT_TYPE_SELECTION_CHANGED events.

GR_EVENT_MOUSE

Name

GR_EVENT_MOUSE — Mouse position event structure

Synopsis

```
typedef struct
{
    GR_EVENT_TYPE   type;
    GR_WINDOW_ID    wid;
    GR_WINDOW_ID    subwid;
    GR_COORD        rootx;
    GR_COORD        rooty;
    GR_COORD        x;
    GR_COORD        y;
```

```
    GR_BUTTON       buttons;
    GR_KEYMOD       modifiers;
} GR_EVENT_MOUSE;
```

Description

The GR_EVENT_MOUSE structure is used by nano-X to report changes in the position of the mouse. When the mouse position changes only one mouse event is sent to a client. The event is sent to the highest window that has selected for the event. If the window's parent has also selected for mouse events, nano-X will not send an additional event for the parent window.

If a window has selected both `GR_EVENT_TYPE_BUTTON_DOWN` and `GR_EVENT_TYPE_BUTTON_UP` events, nano-X will grab the mouse for that window when a mouse button is first pressed down within that window. While the mouse is grabbed, no mouse button or position events will be delivered to any window except the window that nano-X grabbed the mouse for. The mouse will remain grabbed until all of the mouse buttons are released.

Fields

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be either a `GR_EVENT_TYPE_MOUSE_MOTION or a GR_EVENT_TYPE_MOUSE_POSITION type.`

Type	Name	Description
GR_WINDOW_ID	wid	The ID of the window that the mouse event is being sent to. If the mouse has been grabbed then this is the window that nano-X grabbed the mouse for. In this case the mouse may not actually be positioned over the window any longer. The mouse may be over a child window or it may be outside the window that grabbed the mouse.
GR_WINDOW_ID	subwid	The ID of the window that the mouse event occurs in. Generally this field will be the same as *`wid, but in some cases if the mouse event occurs in a decendant of wid, then this field indicates that child window.`*
GR_COORD	rootx	The X coordinate of the mouse pointer relative to the root window.
GR_COORD	rooty	The Y coordinate of the mouse pointer relative to the root window.
GR_COORD	x	The X coordinate of the mouse pointer relative to the window *`wid.`*
GR_COORD	y	The Y coordinate of the mouse pointer relative to the window *`wid.`*
GR_BUTTON	buttons	Indicates the buttons that are being pressed.
GR_KEYMOD	modifiers	Indicates the status of the keyboard modifier keys.

See Also

GR_EVENT, GR_EVENT_BUTTON.

GR_EVENT_SCREENSAVER

Name

GR_EVENT_SCREENSAVER — Screen saver event structure

Synopsis

```
typedef struct
{
    GR_EVENT_TYPE   type;
    GR_BOOL         activate;
} GR_EVENT_SCREENSAVER;
```

Description

The GR_EVENT_SCREENSAVER structure is used to facilitate activation and deactivation of a screen saver. The event is sent to activate a screen saver when the screen saver timer expires with no user input. The event is also sent to deactivate a screen saver if the screen saver is active and some user input is detected.

> **Note:** A bug in version 0.89-pre7 of Microwindows causes the timeout to be multiplied internally by 1000. Therefore when building with 0.89-pre7 you must set the timeout in seconds rather than milli-seconds.

Fields

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be a `GR_EVENT_TYPE_SCREENSAVER`.
GR_BOOL	activate	This field indicates whether the screen saver is being activated or deactivated. If the value is `GR_TRUE then the screen saver should be started. If the value is GR_FALSE then the screen saver should be turned off (screen restored).`

Example

The following example will turn an LCD backlight off when a 60 second screen saver timer expires, and restore the backlight when user input is resumed.

Example 3-1. Using GR_EVENT_SCREENSAVER

```
void setup_screensaver (void)
{
    /* Set a one minute timeout for the LCD backlight */
    /* nano-X BUG: use seconds rather than mS for 0.89-pre7 */
    GrSetScreenSaverTimeout (60 * 1000);
}

void process_screensaver_event (GR_EVENT_SCREENSAVER *event)
{
    if (event->activate)
    {
        /* Turn the LCD backlight off */
        your_platforms_backlight_off();
```

```
    }
    else
    {
        /* Turn the LCD backlight on */
        your_platforms_backlight_on();
    }
}
```

See Also

GR_EVENT, `GrSetScreenSaverTimeout()`.

GR_EVENT_TYPE

Name

`GR_EVENT_TYPE` — Event types

Synopsis

```
typedef int GR_EVENT_TYPE;
```

Description

A GR_EVENT_TYPE enumeration type identifies the type of event that a window is receiving. When a window receives an event, the associated GR_EVENT structure contains a field that specifies the event type.

The following table shows all of the available enumeration values that can be assigned to a GR_EVENT_TYPE variable.

Table 3-1. Event Type Enumerations

Value	Description
GR_EVENT_TYPE_ERROR	This event is sent to the nano-X client when run time errors occur on the nano-X server. It is sent along with a GR_EVENT_ERROR structure.
GR_EVENT_TYPE_NONE	This event type is not used.
GR_EVENT_TYPE_EXPOSURE	This event is sent to the nano-X client when a portion of a window becomes visible and needs to be redrawn. It is sent along with a GR_EVENT_EXPOSURE structure.
GR_EVENT_TYPE_BUTTON_DOWN	This event is sent to the nano-X client when any one or more of the mouse butttons is pressed. It is sent along with a GR_EVENT_BUTTON structure.
GR_EVENT_TYPE_BUTTON_UP	This event is sent to the nano-X client when any one or more of the mouse butttons is released. It is sent along with a GR_EVENT_BUTTON structure.
GR_EVENT_TYPE_MOUSE_ENTER	This event is sent to the nano-X client when the mouse moves onto the window specified within the event. It is sent along with a GR_EVENT_GENERAL structure.

Value	Description
GR_EVENT_TYPE_MOUSE_EXIT	This event is sent to the nano-X client when the mouse moves off of the window specified within the event. It is sent along with a GR_EVENT_GENERAL structure.
GR_EVENT_TYPE_MOUSE_MOTION	This event is sent to the nano-X client when the mouse moves. One of these events is queued for each movement o fthe mouse. Therefore a large number of these events may build up in the event queue. Since none of thes events are thrown away these events will reflect a accurate tracking of the mouse movements. It is sent along with a GR_EVENT_MOUSE structure.
GR_EVENT_TYPE_MOUSE_POSITION	This event is sent to the nano-X client when the mouse moves. These events do NOT build up in the event queue. On each mouse movement the event queue is purged of any events of this type before another event of this type is placed in the queue. This queueing behavior makes this event ideal for realtime response, such as rubber banding or dragging. It is sent along with a GR_EVENT_MOUSE structure.
GR_EVENT_TYPE_KEY_DOWN	This event is sent to the nano-X client when a key on the keyboard is pressed. It is sent along with a GR_EVENT_KEYSTROKE structure.
GR_EVENT_TYPE_KEY_UP	This event is not used in Microwindows revision 0.89pre6.

Value	Description
GR_EVENT_TYPE_FOCUS_IN	This event is sent to the nano-X client when the focus moves to the window specified within the event. It is sent along with a GR_EVENT_GENERAL structure.
GR_EVENT_TYPE_FOCUS_OUT	This event is sent to the nano-X client when the focus moves away from the window specified within the event. It is sent along with a GR_EVENT_GENERAL structure.
GR_EVENT_TYPE_FDINPUT	This event is sent to the nano-X client when data is available for reading on a file descriptor previously registered with the GrRegisterInput() function. It is sent along with a GR_EVENT_FDINPUT structure.
GR_EVENT_TYPE_UPDATE	This event is sent to the nano-X client when an update occurs to the window specified within the event. Window updates occur when the window is moved, resized, mapped, unmapped, activated or destroyed. It is sent along with a GR_EVENT_UPDATE structure.
GR_EVENT_TYPE_CHLD_UPDATE	This event is sent to the nano-X client when an update occurs to a child of the window specified within the event. Window updates occur when the window is moved, resized, mapped, unmapped, activated or destroyed. It is sent along with a GR_EVENT_UPDATE structure.

Value	Description	
GR_EVENT_TYPE_CLOSE_REQ	This event is sent to the nano-X client just before the window specified within the event is closed. It is sent along with a GR_EVENT_GENERAL structure.	
GR_EVENT_TYPE_TIMEOUT	This event is sent to the nano-X client whenever the servers select loop times out. The event is sent regardless of whether it has been enabled by the client. It is sent along with a GR_EVENT_GENERAL structure.	
GR_EVENT_TYPE_SCREENSAVER	This event is sent to the nano-X client whenever the servers screensaver timer times out. It is sent along with a GR_EVENT_SCREENSAVER structure.	
GR_EVENT_TYPE_CLIENT_DATA_REQ		
GR_EVENT_TYPE_CLIENT_DATA		
GR_EVENT_TYPE_SELECTION_CHANGED		

GR_EVENT_UPDATE

Name

GR_EVENT_UPDATE — Window update event structure

Synopsis

```
typedef struct
```

```
{
    GR_EVENT_TYPE   type;
    GR_WINDOW_ID    wid;
    GR_WINDOW_ID    subwid;
    GR_COORD        x;
    GR_COORD        y;
    GR_SIZE         width;
    GR_SIZE         height;
    GR_UPDATE_TYPE  utype;
} GR_EVENT_UPDATE;
```

Description

The GR_EVENT_UPDATE structure is used by nano-X to pass data related to the events `GR_EVENT_TYPE_UPDATE` and `GR_EVENT_TYPE_CHILD_UPDATE`.

Update events are sent to the nano-X client if they have been selected for a window. The update events are "sent to" the window that the event is selected for. `GR_EVENT_TYPE_UPDATE` events are sent to a window when that window is updated. `GR_EVENT_TYPE_CHILD_UPDATE` events are sent to a window when one of it’s child windows is updated. Windows are "updated" when they are moved, resized, mapped, unmapped, activated and destroyed.

Fields

Type	Name	Description
GR_EVENT_TYPE	type	The event type will be `GR_EVENT_TYPE_UPDATE or GR_EVENT_TYPE_CHLD_UPDATE.`

Type	Name	Description
GR_WINDOW_ID	wid	This field is the ID of the window to which the event is being sent. When the event type is `GR_EVENT_TYPE_UPDATE` this is the window that is being updated. When the event type is `GR_EVENT_TYPE_CHILD_UPDATE` then this field indicates one of the updated window's ancestors. The particular ancestor is the window that the event is being sent to.
GR_WINDOW_ID	subwid	This field indicates the window that was updated. If the event type is `GR_EVENT_TYPE_UPDATE` then this field is the same as the *wid field. Otherwise this field is a decendant of wid. In that case this is the actual window that is being updated.*
GR_COORD	x	The updated X position of the window. The field is unused if the update type is `GR_UPDATE_UNMAP`, `GR_UPDATE_UNMAPTEMP` or `GR_UPDATE_ACTIVATE`.
GR_COORD	y	The updated Y position of the window. The field is unused if the update type is `GR_UPDATE_UNMAP`, `GR_UPDATE_UNMAPTEMP` or `GR_UPDATE_ACTIVATE`.

Type	Name	Description
GR_SIZE	width	The updated width of the window. The field is unused if the update type is `GR_UPDATE_UNMAP`, `GR_UPDATE_UNMAPTEMP` or `GR_UPDATE_ACTIVATE`.
GR_SIZE	height	The updated height of the window. The field is unused if the update type is `GR_UPDATE_UNMAP`, `GR_UPDATE_UNMAPTEMP` or `GR_UPDATE_ACTIVATE`.
GR_UPDATE_TYPE	utype	This field indicates the type of update that triggered the event. The update type may be one of: `GR_UPDATE_MAP`, `GR_UPDATE_UNMAP`, `GR_UPDATE_MOVE`, `GR_UPDATE_SIZE`, `GR_UPDATE_UNMAPTEMP`, `GR_UPDATE_ACTIVATE`, `GR_UPDATE_DESTROY`.

See Also

GR_EVENT.

GR_FNCALLBACKEVENT

Name

`GR_FNCALLBACKEVENT` — Call back function prototype

Synopsis

```
typedef void (*GR_FNCALLBACKEVENT)(GR_EVENT *);
```

Description

The GR_FNCALLBACKEVENT type is used to pass the address of a callback function to nano-X. The nano-X library uses callback functions for passing error information to the application (see `GrSetErrorHandler()`) and for driving the applications main event processor (see `GrMainLoop()` or `GrServiceSelect()`).

See Also

`GrSetErrorHandler()`, `GrServiceSelect()`, `GrMainLoop()`.

GR_FONT_ID

Name

`GR_FONT_ID` — Font ID

Synopsis

```
typedef GR_ID  GR_FONT_ID;
```

Description

The GR_FONT_ID type uniquely identifies a nano-X font.

See Also

GR_ID, `GrCreateFont()`.

GR_FONT_INFO

Name

`GR_FONT_INFO` — Font properties

Synopsis

```
typedef struct
{
    int      maxwidth;
    int      height;
    int      baseline;
    int      firstchar;
    int      lastchar;
    int      fixed;
    GR_CHAR  widths[256];
```

```
} GR_FONT_INFO;
```

Description

A GR_FONT_INFO structure contains properties of a nano-X font. The font properties are filled in by the function `GrGetFontInfo()`.

Fields

Type	Name	Description
int	maxwidth	The width of the widest character in the font.
int	height	The height of the font in pixels.
int	baseline	The baseline ascent of the font in pixels.
int	firstchar	The first character in the font.
int	lastchar	The last character in the font.
int	fixed	GR_TRUE if the font is fixed width, GR_FALSE if it is a proportional width font.
GR_CHAR	widths[256]	An array containing the width in pixels of each character in the font.

See Also

`GrGetFontInfo()`, GR_LOGFONT.

GR_FUNC_NAME

Name

GR_FUNC_NAME — Function name string

Synopsis

```
typedef char  GR_FUNC_NAME[25];
```

Description

The GR_FUNC_NAME type holds a 25 byte ASCII string. It is used as a member of the GR_EVENT_ERROR structure.

GR_GC_ID

Name

GR_GC_ID — Graphics context ID

Synopsis

```
typedef GR_ID  GR_GC_ID;
```

Description

The GR_GC_ID type uniquely identifies a nano-X graphics context.

See Also

GR_ID, `GrNewGC()`, `GrCopyGC()`.

GR_GC_INFO

Name

`GR_GC_INFO` — Graphics context information

Synopsis

```
typedef struct
{
    GR_GC_ID        gcid;
    int             mode;
    GR_REGION_ID    region;
    GR_FONT_ID      font;
    GR_COLOR        foreground;
    GR_COLOR        background;
    GR_BOOL         usebackground;
} GR_GC_INFO;
```

Description

The GR_GC_INFO structure is used to retrieve information regarding the current state of a graphics context.

Fields

Type	Name	Description
GR_GC_ID	gcid	The ID of the graphics context.
int	mode	The current drawing mode of the GC.
GR_REGION_ID	region	The current drawing region of the GC.
GR_FONT_ID	font	The current font for drawing text with the GC.
GR_COLOR	foreground	The current foreground color for drawing functions using the GC.
GR_COLOR	background	The current background color for drawing functions using the GC.
GR_BOOL	usebackground	If `GR_TRUE the window background will be drawn when an exposure event occurs. If GR_FALSE the window background will not automatically be drawn.`

The following table shows the drawing modes that are available for use with graphics contexts in nano-X.

Table 3-1. Drawing Modes

Value	Description
`GR_MODE_SET`	When drawing the graphic output will represent the selected forground color.

Value	Description
GR_MODE_XOR	When drawing the graphic output will be th XOR of the GC's foreground color and the current color on the drawable.
GR_MODE_OR	When drawing the graphic output will be th OR of the GC's foreground color and the current color on the drawable.
GR_MODE_AND	When drawing the graphic output will be th AND of the GC's foreground color and the current color on the drawable.
GR_MODE_DRAWMASK	Set bits in this mask correspond to GC mode bits that define drawing style. Clear bits of this mask correspond to GC mode bits that have an extended meaning beyond the drawing style. In this table all of the preceding mode bits define drawing style, all of the following bits have an extended meaning.
GR_MODE_EXCLUDECHILDREN	If this flag is set, then while clipping child windows are excluded from the clip region. Normally the area covered by child windows is clipped when drawing on the parent window. This flag disables the normal clipping action.

See Also

GrGetGCInfo(), GrSetGCMode(), GrSetGCRegion(), GrSetGCFont(), GrSetGCForeground(), GrSetGCBackground(), GrSetGCUseBackground().

GR_ID

Name

`GR_ID` — Generic ID type

Synopsis

```
typedef unsigned int  GR_ID;
```

Description

The GR_ID type uniquely identifies a nano-X object. This type is the base type for all of the other nano-X object type identifiers.

See Also

GR_DRAW_ID, GR_FONT_ID, GR_GC_ID, GR_IMAGE_ID, GR_REGION_ID, GR_WINDOW_ID.

GR_IMAGE_ID

Name

GR_IMAGE_ID — Image ID

Synopsis

```
typedef GR_ID  GR_IMAGE_ID;
```

Description

The GR_IMAGE_ID type uniquely identifies a nano-X image.

See Also

GR_ID, GrLoadImageFromFile().

GR_IMAGE_INFO

Name

GR_IMAGE_INFO — Image properties

Synopsis

```
typedef struct
{
    GR_IMAGE_ID  id;
```

```
    int          width;
    int          height;
    int          planes;
    int          bpp;
    int          pitch;
    int          bytesperpixel;
    int          compression;
    int          palsize;
    GR_PALENTRY  palette[256];
} GR_IMAGE_INFO;
```

Description

A GR_IMAGE_INFO structure contains properties of a nano-X image. The image properties are filled in by the function `GrGetImageInfo()`.

Fields

Type	Name	Description
GR_IMAGE_ID	id	The image ID.
int	width	The width of the image in pixels.
int	height	The height of the image in pixels.
int	planes	The number of color planes in the image.
int	bpp	The number if bits per pixel in the image.
int	pitch	The number of bytes per line in the image.
int	bytesperpixel	The number of bytes per pixel in the image.
int	compression	The compression algorithm used in the image.

Type	Name	Description
int	palsize	The number pallete entries used by the image.
GR_PALENTRY	palette[256]	The image's color palette.

See Also

`GrGetImageInfo()`, GR_IMAGE_HDR.

GR_IMAGE_HDR

Name

`GR_IMAGE_HDR` — Image header

Synopsis

```
typedef struct
{
    int             width;
    int             height;
    int             planes;
    int             bpp;
    int             pitch;
    int             bytesperpixel;
    int             compression;
    int             palsize;
    GR_COLOR        transcolor;
```

```
    GR_PALENTRY   *palette;
    unsigned char *imagebits;
} GR_IMAGE_HDR;
```

Description

A GR_IMAGE_HDR structure defines a nano-X image. This structure is the image header. It in turn points to a palette array and a bitmap array that combined contain the image data.

> **Note:** You use this structure with the `GrDrawImageBits()` function. The utility application convbmp that comes with Microwindows will build GR_IMAGE_HDR structures that may be compiled into your application.

Fields

Type	Name	Description
int	width	The width of the image in pixels.
int	height	The height of the image in pixels.
int	planes	The number of color planes in the image.
int	bpp	The number if bits per pixel in the image.
int	pitch	The number of bytes per line in the image.
int	bytesperpixel	The number of bytes per pixel in the image.
int	compression	The compression algorithm used in the image.

Type	Name	Description
int	palsize	The number pallete entries used by the image.
GR_COLOR	transcolor	This field defines a color that if contained in the image will appear transparent. When the image is drawn any pixels that are of this color are not drawn, thus letting the existing screen iage through. Set this field to -1 if no transparent color is desired.
GR_PALENTRY*	palette	A pointer to the image's color palette.
unsigned char*	imagebits	A pointer to a bitmap array containing the image data.

See Also

`GrDrawImageBits()`, GR_IMAGE_INFO.

GR_KEY

Name

`GR_KEY` — Keyboard key codes

Synopsis

```
typedef unsigned short GR_KEY;
```

Description

A GR_KEY type indicates the value of a keyboard keystroke. The 16 bit value generally contains an ASCII character or a Unicode-16 value. Other keystroke values appear in the following table.

Table 3-1. Key Codes

Value	Description
MWKEY_UNKNOWN	Unknown key.
MWKEY_BACKSPACE	**Back Space key.**
MWKEY_TAB	**Tab key.**
MWKEY_ENTER	**Enter key.**
MWKEY_ESCAPE	**Esc key.**
MWKEY_LEFT	Left arrow key.
MWKEY_RIGHT	Right arrow key.
MWKEY_UP	Up arrow key.
MWKEY_DOWN	Down arrow key.
MWKEY_INSERT	**Insert key.**
MWKEY_DELETE	**Delete key.**
MWKEY_HOME	**Home key.**
MWKEY_END	**End key.**
MWKEY_PAGEUP	**Page Up key.**
MWKEY_PAGEDOWN	**Page Down key.**
MWKEY_KP0	Numeric keypad **0 key.**
MWKEY_KP1	Numeric keypad **1 key.**
MWKEY_KP2	Numeric keypad **2 key.**
MWKEY_KP3	Numeric keypad **3 key.**
MWKEY_KP4	Numeric keypad **4 key.**

Value	Description
MWKEY_KP5	Numeric keypad **5 key.**
MWKEY_KP6	Numeric keypad **6 key.**
MWKEY_KP7	Numeric keypad **7 key.**
MWKEY_KP8	Numeric keypad **8 key.**
MWKEY_KP9	Numeric keypad **9 key.**
MWKEY_KP_PERIOD	Numeric keypad **. key.**
MWKEY_KP_DIVIDE	Numeric keypad **/ key.**
MWKEY_KP_MULTIPLY	Numeric keypad *** key.**
MWKEY_KP_MINUS	Numeric keypad **- key.**
MWKEY_KP_PLUS	Numeric keypad **+ key.**
MWKEY_KP_ENTER	Numeric keypad **Enter key.**
MWKEY_KP_EQUALS	Numeric keypad **= key.**
MWKEY_F1	**F1 key.**
MWKEY_F2	**F2 key.**
MWKEY_F3	**F3 key.**
MWKEY_F4	**F4 key.**
MWKEY_F5	**F5 key.**
MWKEY_F6	**F6 key.**
MWKEY_F7	**F7 key.**
MWKEY_F8	**F8 key.**
MWKEY_F9	**F9 key.**
MWKEY_F10	**F10 key.**
MWKEY_F11	**F11 key.**
MWKEY_F12	**F12 key.**
MWKEY_NUMLOCK	**Num Lock key.**
MWKEY_CAPSLOCK	**Caps Lock key.**

Value	Description
MWKEY_SCROLLOCK	**Scroll Lock key.**
MWKEY_LSHIFT	Left **Shift key.**
MWKEY_RSHIFT	Right **Shift key.**
MWKEY_LCTRL	Left **Ctrl key.**
MWKEY_RCTRL	Right **Ctrl key.**
MWKEY_LALT	Left **Alt key.**
MWKEY_RALT	Right **Alt key.**
MWKEY_LMETA	Left window key.
MWKEY_RMETA	Right window key.
MWKEY_ALTGR	**AlrGr key.**
MWKEY_PRINT	**PrintScrn key.**
MWKEY_SYSREQ	**SysReq key.**
MWKEY_PAUSE	**Pause key.**
MWKEY_BREAK	**Break key.**
MWKEY_QUIT	**Quit key.**
MWKEY_MENU	**Menu key.**
MWKEY_REDRAW	**ReDraw key.**
MWKEY_RECORD	**Record key.**
MWKEY_PLAY	**Play key.**
MWKEY_CONTRAST	**Contrast key.**
MWKEY_BRIGHTNESS	**Brightness key.**
MWKEY_SELECTUP	Select up key.
MWKEY_SELECTDOWN	Select down key.
MWKEY_ACCEPT	**Accept key.**
MWKEY_CANCEL	**Cancel key.**
MWKEY_APP1	**App1 key.**

Value	Description
MWKEY_APP2	**App2 key.**
MWKEY_FIRST	**First key.**
MWKEY_LAST	**Last key.**

Not all of the key codes are available on all architectures. The constant MWKEY_NONASCII_MASK can be OR'd with a key code to determine if the key code is outside tha ASCII character range.

See Also

GR_KEYMOD, GR_EVENT_KEYSTROKE.

GR_KEYMOD

Name

GR_KEYMOD — Keyboard modifier key codes

Synopsis

```
typedef unsigned int GR_KEYMOD;
```

Description

A GR_KEYMOD type is a bitwise OR combination of one or more of the following flags. The set flags indicate keyboard modifier keys that are being pressed.

Table 3-1. Modifier Key Codes

Value	Description
MWKMOD_NONE	Indicates no modifier keys are pressed.
MWKMOD_LSHIFT	Indicates the left **Shift key.**
MWKMOD_RSHIFT	Indicates the right **Shift key.**
MWKMOD_SHIFT	Bitwise OR of the left and right **Shift keys.**
MWKMOD_LCTRL	Indicates the left **Ctrl key.**
MWKMOD_RCTRL	Indicates the right **Ctrl key.**
MWKMOD_CTRL	Bit wise OR of the left and right **Ctrl keys.**
MWKMOD_LALT	Indicates the left **Alt key.**
MWKMOD_RALT	Indicates the right **Alt key.**
MWKMOD_ALT	Bitwise OR if the left and right **Alt key.**
MWKMOD_LMETA	Indicates the left **window key.**
MWKMOD_RMETA	Indicates the right **window key.**
MWKMOD_META	Bitwise OR of the left and right **window keys.**
MWKMOD_NUM	Indicates the **Num Lock key.**
MWKMOD_CAPS	Indicates the**Caps Lock key.**
MWKMOD_ALTGR	Indicates the **AltGr key.**

See Also

GR_KEY, GR_EVENT_BUTTON, GR_EVENT_KEYSTROKE, GR_EVENT_MOUSE.

GR_LOGFONT

Name

GR_LOGFONT — Font properties

Synopsis

```
typedef struct
{
    long     lfHeight;
    long     lfWidth;
    long     lfEscapement;
    long     lfOrientation;
    long     lfWeight;
    GR_CHAR  lfItalic;
    GR_CHAR  lfUnderline;
    GR_CHAR  lfStrikeOut;
    GR_CHAR  lfCharSet;
    GR_CHAR  lfOutPrecision;
    GR_CHAR  lfClipPrecision;
    GR_CHAR  lfQuality;
    GR_CHAR  lfRoman;
    GR_CHAR  lfSerif;
    GR_CHAR  lfSansSerif;
    GR_CHAR  lfModern;
    GR_CHAR  lfMonospace;
    GR_CHAR  lfProportional;
    GR_CHAR  lfOblique;
    GR_CHAR  lfSmallCaps;
    GR_CHAR  lfPitch;
    char     lfFaceName[MWLF_FACESIZE];
} GR_LOGFONT;
```

Description

A GR_LOGFONT structure is a logical font descriptor it is used to specify the desired font characteristics when creating a font. An application passes one of these structures into the `GrCreateFont()` function when a font is created.

Several macros are defined in `mwtypes.h` to help build the GR_LOGFONT structure. These macros are shown below.

Fields

Type	Name	Description
long	lfHeight	The font height in pixels.
long	lfWidth	The font width in pixels.
long	lfEscapement	The rotation of the font in tenths of a degree.
long	lfOrientation	This field is not currently used.
long	lfWeight	The font weight. See below
GR_CHAR	lfItalic	`GR_TRUE for an italic font, otherwise GR_FALSE.`
GR_CHAR	lfUnderline	`GR_TRUE for an underlined font, otherwise GR_FALSE.`
GR_CHAR	lfStrikeOut	This field is currently not used.
GR_CHAR	lfCharSet	The font character set. See below
GR_CHAR	lfOutPrecision	Font type selection. See below
GR_CHAR	lfClipPrecision	This field is currently not used.
GR_CHAR	lfQuality	This field is currently not used.
GR_CHAR	lfRoman	`GR_TRUE for Roman (upright) letters, otherwise GR_FALSE.`

Type	Name	Description
GR_CHAR	lfSerif	GR_TRUE for a serifed font, otherwise GR_FALSE.
GR_CHAR	lfSansSerif	GR_TRUE for a sans-serif font, otherwise GR_FALSE.
GR_CHAR	lfModern	GR_TRUE for a modern font, otherwise GR_FALSE.
GR_CHAR	lfMonospace	GR_TRUE for a monsspaced font, otherwise GR_FALSE.
GR_CHAR	lfProportional	GR_TRUE for a proportionally spaced, otherwise GR_FALSE.
GR_CHAR	lfOblique	GR_TRUE for an oblique font, otherwise GR_FALSE.
GR_CHAR	lfSmallCaps	GR_TRUE for a small caps font, otherwise GR_FALSE.
GR_CHAR	lfPitch	The font pitch. See below
char	lfFaceName[]	Zero terminated ASCII string containing the font face name.

Field Enumerations

The following paragraphs list the enumeration values that may be used with the fields of the GR_LOGFONT structure.

Enumerations for the *lfOutPrecision* field.

MWLF_TYPE_DEFAULT	MWLF_TYPE_SCALED	MWLF_TYPE_RASTER
MWLF_TYPE_TRUETYPE	MWLF_TYPE_ADOBE	

Enumerations for the *lfWeight* field.

MWLF_WEIGHT_DEFAULT	MWLF_WEIGHT_MEDIUM
MWLF_WEIGHT_THIN	MWLF_WEIGHT_DEMIBOLD
MWLF_WEIGHT_EXTRALIGHT	MWLF_WEIGHT_BOLD
MWLF_WEIGHT_LIGHT	MWLF_WEIGTH_EXTRABOLD
MWLF_WEIGHT_NORMAL	MWLF_WEIGHT_BLACK
MWLF_WEIGHT_REGULAR	

Enumerations for the *lfCharSet* field.

MWLF_CHARSET_ANSI	MWLF_CHARSET_DEFAULT
MWLF_CHARSET_UNICODE	MWLF_CHARSET_OEM

Enumerations for the *lfPitch* field.

MWLF_PITCH_DEFAULT	MWLF_PITCH_NORMAL
MWLF_PITCH_ULTRACONDENSED	MWLF_PITCH_SEMIEXPANDED
MWLF_PITCH_EXTRACONDENSED	MWLF_PITCH_EXPANDED
MWLF_PITCH_CONDENSED	MWLF_PITCH_EXTRAEXPANDED
MWLF_PITCH_SEMICONDENSED	MWLF_PITCH_ULTRAEXPANDED

Built in font face names for the *lfFaceName* field.

GR_FONT_SYSTEM_VAR	GR_FONT_OEM_FIXED
GR_FONT_GUI_VAR	GR_FONT_SYSTEM_FIXED

Macros

The following macros set the

Table 3-1. GR_LOGFONT Macros

Name	Description
`MWLF_Clear (GR_LOGFONT *lf)`	This macro sets the size fields of the structure to zero, the weight to `MWLF_WEIGHT_REGULAR`, all of the flags to `GR_FALSE` and the face name to an empty string.
`MWLF_SetBold (GR_LOGFONT *lf)`	This macro modifies the GR_LOGFONT so that it describes a bold font. The *`lfWeight is set to the constant value MWLF_WEIGHT_BOLD.`*
`MWLF_SetRegular (GR_LOGFONT *lf)`	This macro modifies the GR_LOGFONT so that it describes a non-bold font. The *`lfWeight is set to the constant value MWLF_WEIGHT_REGULAR.`*
`MWLF_SetItalics (GR_LOGFONT *lf)`	This macro modifies the GR_LOGFONT so that it describes an italic font. The *`lfItalic is set to GR_TRUE, the lfOblique is set to GR_FALSE and the lfRoman is set to GR_FALSE.`*
`MWLF_SetRoman (GR_LOGFONT *lf)`	This macro modifies the GR_LOGFONT so that it describes a roman font. The *`lfItalic is set to GR_FALSE, the lfOblique is set to GR_FALSE and the lfRoman is set to GR_TRUE.`*

See Also

`GrCreateFont()`, GR_FONT_INFO.

GR_PALENTRY

Name

`GR_PALENTRY` — Palette entry

Synopsis

```
typedef struct
{
    unsigned char  r;
    unsigned char  g;
    unsigned char  b;
} GR_PALENTRY;
```

Description

The GR_PALENTRY structure is the core element of a color palette in the nano-X library.

Fields

Type	Name	Description
unsigned char	r	The red component of the color.
unsigned char	g	The green component of the color.
unsigned char	b	The blue component of the color.

See Also

GR_PALETTE, GR_IMAGE_INFO.

GR_PALETTE

Name

GR_PALETTE — Color palette

Synopsis

```
typedef struct
{
    GR_COUNT       count;
    GR_PALENTRY    palette[256];
} GR_PALETTE;
```

Description

The GR_PALETTE structure is used to describe an array of colors that the system will render.

Fields

Type	Name	Description
GR_COUNT	count	The number of palette entrys in the *palette array.*
GR_PALENTRY	palette	An array of palette colors.

GR_PIXELVAL

Name

GR_PIXELVAL — Hardware dependent color value

Synopsis

```
#if MWPIXEL_FORMAT == MWPF_TRUECOLOR565
typedef unsigned short GR_PIXELVAL;
#else
  #if MWPIXEL_FORMAT == MWPF_TRUECOLOR332
  typedef unsigned char GR_PIXELVAL;
  #else
    #if MWPIXEL_FORMAT == MWPF_PALETTE
    typedef unsigned char GR_PIXELVAL;
    #else
      typedef unsigned long GR_PIXELVAL;
```

```
    #endif
  #endif
#endif
```

Description

The GR_PIXELVAL type is a hardware dependent color value it is typically used for device dependent image storage. The size of this value is dependent on the configuration settings that were used to build the Microwindows libraries.

GR_POINT

Name

`GR_POINT` — Point structure

Synopsis

```
typedef struct
{
    GR_COORD     x;
    GR_COORD     y;
} GR_POINT;
```

Description

A GR_POINT structure defines the position of a single pixel point on the screen or within a drawable.

Fields

Type	Name	Description
GR_COORD	x	The X coordinate of the point.
GR_COORD	y	The Y coordinate of the point.

GR_RECT

Name

GR_RECT — Rectangle structure

Synopsis

```
typedef struct
{
    GR_COORD     x;
    GR_COORD     y;
    GR_SIZE      width;
    GR_SIZE      height;
} GR_RECT;
```

Description

A GR_RECT structure defines the size and position of a rectangle.

Fields

Type	Name	Description
GR_COORD	x	The X coordinate of the upper left corner of the rectangle.
GR_COORD	y	The Y coordinate of the upper left corner of the rectangle.
GR_SIZE	width	The width, in pixels, of the rectangle.
GR_SIZE	height	The height, in pixels, of the rectangle.

GR_REGION_ID

Name

`GR_REGION_ID` — Region ID

Synopsis

```
typedef GR_ID  GR_REGION_ID;
```

Description

The GR_REGION_ID type uniquely identifies a nano-X region.

See Also

GR_ID, `GrNewRegion()`, `GrNewPolygonRegion()`.

GR_SCANCODE

Name

`GR_SCANCODE` — OEM keyboard scancode

Synopsis

```
typedef unsigned char  GR_SCANCODE;
```

Description

The GR_SCANCODE type holds a device dependent OEM keyboard scancode value.

GR_SCREEN_INFO

Name

GR_SCREEN_INFO — Screen properties

Synopsis

```
typedef struct
{
    GR_COORD   rows;
    GR_COORD   cols;
    GR_SIZE    xdpcm;
    GR_SIZE    ydpcm;
    GR_COUNT   planes;
    GR_COUNT   bpp;
    long       ncolors;
    GR_COUNT   fonts;
    GR_BUTTON  buttons;
    GR_KEYMOD  modifiers;
    int        pixtype;
    GR_COORD   xpos;
    GR_COORD   ypos;
    GR_SIZE    vs_width;
    GR_SIZE    vs_height;
    GR_SIZE    ws_width;
    GR_SIZE    ws_height;
} GR_SCREEN_INFO;
```

Description

A GR_SCREEN_INFO structure lets your application determine screen properties at run time. The structure is

Fields

Type	Name	Description
GR_COORD	rows	The number of rows of pixels on the screen.
GR_COORD	cols	The number of columns of pixels on the screen.
GR_SIZE	xdpcm	The number of pixels (dots) per centimeter along the X axis of the screen.
GR_SIZE	ydpcm	The number of pixels (dots) per centimeter along the Y axis of the screen.
GR_COUNT	planes	The number of color planes in the graphics hardware.
GR_COUNT	bpp	The number of bits per pixel in the graphics hardware.
long	ncolors	The number of colors supported by the hardware.
GR_COUNT	fonts	The number of built-in fonts.
GR_BUTTON	buttons	This field indicates the buttons that are available on the system’s pointing device. For a touch screen device only a left button is available, for a GPM mouse three butons are available.
GR_KEYMOD	modifiers	This field indicates the modifier keys that are available on the system’s keyboard device.

Type	Name	Description
int	pixtype	The screen drivers native pixel format. See below for a list of the available pixel formats.
GR_COORD	xpos	The current position of the mouse along the X axis.
GR_COORD	ypos	The current position of the mouse along the Y axis.
GR_SIZE	vs_width	
GR_SIZE	vs_height	
GR_SIZE	ws_width	
GR_SIZE	ws_height	

Pixel Formats

The following table lists the possible pixel format values that may be returned in the GR_SCREEN_INFO structure. There are two pseudo pixel formats. These formats will never be returned from a screen driver, but they are be used as a data type with the `GrArea()` function.

Pixel Format	Description
`MWPF_RGB`	This psuedo format is used as a conversion specifier when working with 32 bit RGB format pixel colors.
`MWPF_PIXELVAL`	This psuedo format is used as a *no conversion specifier when working with GR_PIXELVAL pixel colors.*
`MWPF_PALETTE`	Palettized pixel color format.
`MWPF_TRUECOLOR0888`	Packed 32 bit 0/8/8/8 true color format.
`MWPF_TRUECOLOR888`	Packed 24 bit 8/8/8 truecolor format.
`MWPF_TRUECOLOR565`	Packed 16 bit 5/6/5 truecolor format.

Pixel Format	Description
MWPF_TRUECOLOR555	Packed 16 bit 0/5/5/5 truecolor format.
MWPF_TRUECOLOR332	Packed 8 bit 3/3/2 truecolor format.

See Also

GrGetScreenInfo(), GR_WINDOW_INFO.

GR_SIZE

Name

GR_SIZE — Graphic object size

Synopsis

```
typedef int  GR_SIZE;
```

Description

The GR_SIZE type is typically used to specify the width or height of a graphic object, such as a rectangle.

GR_TIMEOUT

Name

`GR_TIMEOUT` — Time out value

Synopsis

```
typedef unsigned long  GR_TIMEOUT;
```

Description

The GR_TIMEOUT type represents time in milli-second increments.

A GR_TIMEOUT value is used by the function `GrGetNextEventTimeout()` to specify a maximum amount of time to block while waiting for the next nano-X event from the event queue.

A GR_TIMEOUT value is used in the GR_EVENT_BUTTON structure to specify the absolute time of the mouse button event.

GR_UPDATE_TYPE

Name

`GR_UPDATE_TYPE` — Window update event, event subtypes

Synopsis

```
typedef int GR_UPDATE_TYPE;
```

Description

A GR_UPDATE_TYPE enumeration type identifies the reason for a window receiving an update event. When a window receives a `GR_EVENT_TYPE_UPDATE` or a `GR_EVENT_TYPE_CHILD_UPDATE` event the corresponding GR_EVENT_UPDATE structure will contain a field of this kind that specifies the update type.

The following table shows all of the available enumeration values that can be assigned to a GR_UPDATE_TYPE variable.

Table 3-1. Update Enumerations

Value	Description
`GR_UPDATE_MAP`	Indicates that the window has been mapped.
`GR_UPDATE_UNMAP`	Indicates that the window has been unmapped.
`GR_UPDATE_MOVE`	Indicates that the window has been moved.
`GR_UPDATE_SIZE`	Indicates that the window has been resized.
`GR_UPDATE_UNMAPTEMP`	Indicates that the window has been temporarily unmapped. A window is temporarily unmapped while it is moved, resized or reparented.
`GR_UPDATE_ACTIVATE`	Indicates that the window has been activated.

Value	Description
GR_UPDATE_DESTROY	Indicates that the window has been destroyed.

GR_WINDOW_ID

Name

GR_WINDOW_ID — Window ID

Synopsis

```
typedef GR_ID  GR_WINDOW_ID;
```

Description

The GR_WINDOW_ID type uniquely identifies a nano-X window or pixmap. The value `0` is an illegal value for a window ID.

GR_ROOT_WINDOW_ID

Nano-X defines a constant window ID GR_ROOT_WINDOW_ID which is always valid. This window ID indicates the *ROOT* window of the nano-X server. The *ROOT* window is the eldest ancestor of all other windows.

See Also

GR_ID, `GrNewWindow()`, `GrNewWindowEx()`, `GrNewPixmap()`, `GrNewInputWindow()`, `GrNewPixmapFromData()`.

GR_WINDOW_INFO

Name

`GR_WINDOW_INFO` — Retrieve window properties

Synopsis

```
typedef struct
{
    GR_WINDOW_ID   wid;
    GR_WINDOW_ID   parent;
    GR_WINDOW_ID   child;
    GR_WINDOW_ID   sibling;
    GR_BOOL        inputonly;
    GR_BOOL        mapped;
    GR_COUNT       unmapcount;
    GR_COORD       x;
    GR_COORD       y;
    GR_SIZE        width;
    GR_SIZE        height;
    GR_SIZE        bordersize;
    GR_COLOR       bordercolor;
    GR_COLOR       background;
    GR_EVENT_MASK  eventmask;
    GR_WM_PROPS    props;
```

```
} GR_WINDOW_INFO;
```

Description

This stucture is used in conjuncion with the `GrGetWindowInfo()` function to return information about a window's current properties.

Fields

Type	Name	Description
GR_WINDOW_ID	wid	The window ID of the window described in this structure, or `0 if the window passed to GrGetWindowInfo() is invalid.`
GR_WINDOW_ID	parent	The window ID of this window's parent window.
GR_WINDOW_ID	child	The window ID of this window's first child window. All of this window's child windows can be determined by obtaining window information on the first child, then the first child's next sibling, then that child's next sibling, etc.. This field will be zero if the window has no children.

Type	Name	Description
GR_WINDOW_ID	sibling	The window ID of this windows's next sibling window. All child windows, of a particular parent, form a singly linked list. This field indicates the next child window in the list. This field will be zero if the window has no siblings, or is the last sibling in the linked list.
GR_BOOL	inputonly	This field is `GR_TRUE if the window is an input only window.`
GR_BOOL	mapped	This field is `GR_TRUE if the window is mapped (visible).`
GR_COUNT	unmapcount	The depth of unmapping for this window. When zero this window is visible. Each time the window is unmapped this field will increase, each time the window is mapped this field will decrement.
GR_COORD	x	The X coordinate of the uper left corner of the window relative to the screen.
GR_COORD	y	The Y coordinate of the uper left corner of the window relative to the screen.
GR_SIZE	width	The width of the window.
GR_SIZE	height	The height of the window.
GR_SIZE	bordersize	The width of the window's border.
GR_COLOR	bordercolor	The color of the window's border.
GR_COLOR	background	The window's background color.
GR_EVENT_MASK	eventmask	The window's event mask. The value of this field indicates all events that the window is selected to receive (see GrSelectEvents()).

Type	Name	Description
GR_WM_PROPS	props	The window's window manager properties.

See Also

`GrGetWindowInfo()`, GR_SCREEN_INFO.

GR_WM_PROPS

Name

`GR_WM_PROPS` — Window manager properties

Synopsis

```
typedef unsigned long GR_WM_PROPS;
```

Description

A GR_WM_PROPS type is a bitwise OR combination of one or more of the following window manger property flags.

Table 3-1. Window Manager Properties

Value	Description
GR_WM_PROPS_NOBACKGROUND	Do not draw the window's background.
GR_WM_PROPS_NOFOCUS	Do not allow focus to be set to this window.
GR_WM_PROPS_NOMOVE	Do not allow the user to move this window.
GR_WM_PROPS_NORAISE	Do not allow the user to raise this window.
GR_WM_PROPS_NODECORATE	Do not redecorate the window.
GR_WM_PROPS_NOAUTOMOVE	Do not move the window on the first mapping.
GR_WM_PROPS_NOAUTORESIZE	Do not resize the window on the first mapping.
GR_WM_PROPS_APPWINDOW	Leave the appearance up to the window manager.
GR_WM_PROPS_APPMASK	This flags masks all of tha appearance specific flags. These include GR_WM_PROPS_BORDER, GR_WM_PROPS_APPFRAME, GR_WM_PROPS_CAPTION, GR_WM_PROPS_CLOSEBOX,
GR_WM_PROPS_BORDER	Give the window a single line border.
GR_WM_PROPS_APPFRAME	Give the window a 3D application frame. This flag overrides the GR_WM_PROPS_BORDER flag.
GR_WM_PROPS_CAPTION	Give the window a title bar.
GR_WM_PROPS_CLOSEBOX	Give the window a close box.
GR_WM_PROPS_MAXIMIZE	The window is maximized.

GR_WM_PROPERTIES

Name

GR_WM_PROPERTIES — Window manager property configuration

Synopsis

```
typedef struct
{
    GR_WM_PROPS     flags;
    GR_WM_PROPS     props;
    GR_CHAR        *title;
    GR_COLOR        background;
    GR_SIZE         bordersize;
    GR_COLOR        bordercolor;
} GR_WM_PROPERTIES;
```

Description

A GR_WM_PROPERTIES structure is used to set and get a window's window manager properties.

Fields

Type	Name	Description

Type	Name	Description
GR_WM_PROPS	flags	These flags indicate which fields within this structure have significance when this structure is used with the function `GrSetWMProperties()`. NOTE: This variable has nothing in common with a GR_WM_PROPS type, except that it consumes the same amount of memory. See the table below for the meaning of the bits within this field.
GR_WM_PROPS	props	The window manager property flags.
GR_CHAR *	title	The text that appears on the window title bar.
GR_COLOR	background	The color of the window background.
GR_SIZE	bordersize	The width of the window border.
GR_COLOR	bordercolor	The color of the window border.

Table 3-1. GR_WM_PROPERTIES Flags

Value	Description
`GR_WM_FLAGS_PROPS`	The *`props field is set.`*
`GR_WM_FLAGS_TITLE`	The *`title field is set.`*
`GR_WM_FLAGS_BACKGROUND`	The *`background field is set.`*
`GR_WM_FLAGS_BORDERSIZE`	The *`bordersize field is set.`*
`GR_WM_FLAGS_BORDERCOLOR`	The *`bordercolor field is set.`*

See Also

`GrSetWMProperties()`, `GrGetWMProperties()`, `GrSetWindowBackgroundColor()`, `GrSetWindowBorderSize()`,

`GrSetWindowBorderColor()`, `GrSetWindowTitle()`.