XBasic

Program Development Environment
(PDE)

GraphicsDesigner

Programmer Guide
Programmer Reference

Revision 0.0020
January 10, 1996

Table of Contents

Overview

GraphicsDesigner

To draw graphics, programs call functions in *GraphicsDesigner*, a built-in function library. Programs can also monitor keyboard and mouse activity by processing messages from the general purpose message queue built into *GraphicsDesigner*.

GraphicsDesigner is the basis for all graphics, from simple line drawings to sophisticated graphical user interfaces. For example, *GuiDesigner*, the interactive GUI design tool built into the program development environment, is a program that calls *GraphicsDesigner* functions.

All *GraphicsDesigner* functions begin with **`Xgr`**, as in:

```
XgrCreateWindow()
XgrDestroyWindow()
XgrCreateGrid()
XgrDestroyGrid()
XgrProcessMessage()
  ... etc ...
```

Windows

Graphics is displayed in graphics *windows*, each of which is a rectangular area on the display with a unique **`window`** number.

`window`

Programs can create any number of windows. To create a window, a program calls:

```
XgrCreateWindow ( @window, winType, x, y, w, h, &winFunc(), disp$ )
```

`XgrCreateWindow()` returns a unique **`window`** number to identify each window it creates, or **`0`** if for any reason no window was created.

Window Types

Windows can have attributes like a title bar, a resize border, a minimize button or maximize button, etc. A particular combination of the following window type attribute bits is called a window type:

```
$$WindowTopMost     - stays above other windows
$$WindowNoSelect    - window is not selected by mouse button events
$$WindowNoFrame     - window has no resize frame
$$WindowResizeFrame - window has a resize frame
$$WindowTitleBar    - window has a title bar to display a window name
$$WindowSystemMenu  - window has a system menu button
$$WindowMinimizeBox - window has a minimize button
$$WindowMaximizeBox - window has a maximize button
```

Some window systems do not support individual selection of the window features shown above, so windows may display more or less features than requested.

Grids

Programs do not draw directly into windows. Programs create one or more *grids* in each window, each of which is a rectangular area with its own coordinate systems and attributes like size, background color, drawing color, drawpoints, etc.

The attribute settings in each grid are private, so graphics operations in one grid have no effect on other grids. No matter how many grids a program creates, each grid is independent.

To create a grid, programs call:

```
XgrCreateGrid ( @grid, gridType, x, y, w, h, window, parent, func )

  grid      - unique grid number returned to identify the grid
  gridType  - number that specifies the grid type of the grid
  x         - horizontal location of left edge of grid
  y         - vertical location of upper edge of grid
  w         - width of grid in pixels
  h         - height of grid in pixels
  window    - window number of window that contains grid
  parent    - parent grid (0 if window is its parent)
  func      - address of grid function that operates grid
```

`grid`

`XgrCreateGrid()` assigns a unique `grid` number to each grid. `grid` is the first argument to most *GraphicsDesigner* functions, so all graphics operations are directed at a single grid, and all other grids are unaffected.

Grid Type

Every grid has a ***grid type***. Simple coordinate grids, suitable for all kinds of graphics, are grid type `0`. Image grids are grid type `1`.

These two grid types are pre-defined by *GraphicsDesigner*, and pure graphics programs need not create others. On the other hand, dozens of grid types exist in GUI programs - see *GuiDesigner*.

Coordinate Systems

GraphicsDesigner supports five coordinate systems:

```
  Display Coordinates - 0,0 is upper left corner of display
  Window Coordinates  - 0,0 is upper left corner of window
  Local Coordinates   - 0,0 is upper left corner of grid
  Grid Coordinates    - specifies corners of grid in pixel units
  Scaled Coordinates  - specifies corners of grid in any units
```

Coordinate Conversion

Conversions between coordinate systems are provided by:

```
XgrConvertDisplayToGrid   (grid, xDisp, yDisp, @xGrid, @yGrid)
XgrConvertDisplayToLocal  (grid, xDisp, yDisp, @x, @y)
XgrConvertDisplayToScaled (grid, xDisp, yDisp, @x#, @y#)
XgrConvertDisplayToWindow (grid, xDisp, yDisp, @xWin, @yWin)

XgrConvertGridToDisplay   (grid, xGrid, yGrid, @xDisp, @yDisp)
XgrConvertGridToLocal     (grid, xGrid, yGrid, @x, @y)
XgrConvertGridToScaled    (grid, xGrid, yGrid, @x#, @y#)
XgrConvertGridToWindow    (grid, xGrid, yGrid, @xWin, @yWin)

XgrConvertLocalToDisplay  (grid, x, y, @xDisp, @yDisp)
XgrConvertLocalToGrid     (grid, x, y, @xGrid, @yGrid)
XgrConvertLocalToScaled   (grid, x, y, @x#, @y#)
XgrConvertLocalToWindow   (grid, x, y, @xWin, @yWin)

XgrConvertScaledToDisplay (grid, x#, y#, @xDisp, @yDisp)
XgrConvertScaledToGrid    (grid, x#, y#, @xGrid, @yGrid)
XgrConvertScaledToLocal   (grid, x#, y#, @x, @y)
XgrConvertScaledToWindow  (grid, x#, y#, @xWin, @yWin)
```

Grid Location Coordinates

The upper-left : lower-right corners of grids are provided by:

```
XgrGetGridBoxDisplay (grid, @x1Disp, @y1Disp, @x2Disp, @y2Disp)
XgrGetGridBoxGrid    (grid, @x1Grid, @y1Grid, @x2Grid, @y2Grid)
XgrGetGridBoxLocal   (grid, @x1, @y1, @x2, @y2)
XgrGetGridBoxScaled  (grid, @x1#, @y1#, @x2#, @y2#)
XgrGetGridBoxWindow  (grid, @x1Win, @y1Win, @x2Win, @y2Win)
```

Drawing Coordinates

Graphics is drawn in any combination of three coordinate systems:

```
Local Coordinates   - XgrDrawPoint (grid, color, x, y)
Grid Coordinates     - XgrDrawPointGrid (grid, color, xGrid, yGrid)
Scaled Coordinates  - XgrDrawPointScaled (grid, color, x#, y#)
```

GraphicsDesigner includes a complete set of drawing functions for local coordinates, grid coordinates, and scaled coordinates.

For example, a line is drawn from the upper-left to lower-right corner of the following grid by all three of the line drawing functions:

```
XgrCreateWindow ( @win, winType, 300, 300, 200, 200, &winFunc(), "" )
XgrCreateGrid ( @grid, gridType, 10, 10, 100, 100, win, 0, gfunc )
XgrSetGridBoxGrid ( grid, -49, 50, 50, -49 )
XgrSetGridBoxScaled ( grid, -320, +85, +14495, -40 )

XgrDrawLine ( grid, $$Red, 0, 0, 99, 99 )
XgrDrawLineGrid ( grid, $$Green, -49, 50, 50, -49 )
XgrDrawLineScaled ( grid, $$Blue, -320, +85, +14495, -40 )
```

Display Coordinates

Display coordinates are the natural coordinates of display screens.

The display coordinates of the upper-left corner of every display is fixed at **`(xDisp,yDisp) = (0,0)`**.

Display coordinates **`(xDisp,yDisp)`** increase by 1 unit per pixel rightward and downward, and cannot be offset, inverted or scaled.

Window Coordinates

Window coordinates are the natural coordinates of windows.

The window coordinates of the upper-left corner of every window is fixed at **`(xWin,yWin) = (0,0)`**.

Window coordinates **`(xWin,yWin)`** increase by 1 unit per pixel rightward and downward and cannot be offset, inverted or scaled.

Local Coordinates

Local coordinates are the natural coordinates of grids.

The local coordinates of the upper-left corner of every grid is fixed at **`(x,y) = (0,0)`**.

Local coordinates **`(x,y)`** increase by 1 unit per pixel rightward and downward and cannot be offset, inverted or scaled.

Grid Coordinates

Grid coordinates can be offset and/or inverted.

Consider a 100x100 pixel grid created by:

```
XgrCreateGrid ( @grid, 0, 200, 200, 100, 100, window, 0, 0 )
```

The upper-left : lower-right corners of this grid are located at window coordinates **(200,200:299,299)**, local coordinates **(0,0:99,99)**, and grid coordinates **(0,0:99,99)**.

When a grid is created, its grid coordinates and local coordinates are the same, **(0,0:w-1,h-1)**, where **w** = width and **h** = height in pixels.

But **XgrSetGridBoxGrid (grid, x1Grid, y1Grid, x2Grid, y2Grid)** can redefine the grid coordinates of the upper-left : lower-right corners of a grid to offset its grid coordinates horizontally and/or vertically from its local coordinates, as well as invert horizontally and/or vertically.

But grid coordinates are always measured in 1 pixel units, and **XgrSetGridBoxGrid()** cannot move grids or alter grid coordinates scale. So any time **XgrSetGridBoxGrid()** is called with **x2Grid** and/or **y2Grid** arguments that are not consistent with grid width and/or height, **x2Grid** and/or **y2Grid** are computed as follows:

```
IF (x1Grid <= x2Grid) THEN
  x2Grid = x1Grid + (width - 1)
ELSE
  x2Grid = x1Grid - (width - 1)
END IF

IF (y1Grid <= y2Grid) THEN
  y2Grid = y1Grid + (height - 1)
ELSE
  y2Grid = y1Grid - (height - 1)
END IF
```

Scaled Coordinates

Scaled coordinates can be offset and/or inverted and/or scaled.

A pair of floating-point scaled coordinates can be assigned to the upper-left : lower-right corners of every grid. For example, scaled coordinates **(-320,+85:+14495,-40)** could be assigned to the **(upper-left:lower-right)** corners of the grid by:

```
XgrSetGridBoxScaled ( grid, -320, 85, 14495, -40 )
```

The corners of this grid are now:

```
Coordinates       Upper-Left      Lower-Right

Local                  (0, 0)          (99, 99)
Scaled           (-320, +85)    (+14495, -40)
```

Grid Box

The rectangle defined by the upper-left and lower-right corners of a grid is its *grid box*. Drawing to a grid is confined to the grid-box.

Example

The following program segment creates a window and grid, clears the grid to black, then draws red, green, and blue lines between its upper-left and lower-right corners.

```
  XgrCreateWindow ( @win, 0, 0, 0, 600, 400, 0, "" )
  XgrCreateGrid ( @grid, 0, 200, 200, 100, 100, win, 0, 0 )
  XgrSetGridBoxGrid ( grid, -99, -99, 0, 0)
  XgrSetGridBoxScaled ( grid, -320, +85, +14995, -40 )
'
  XgrClearGrid ( grid, $$Black )
  XgrDrawLine ( grid, $$Red, 0, 0, 99, 99 )
  XgrDrawLineGrid ( grid, $$Green, -99, -99, 0, 0 )
  XgrDrawLineScaled ( grid, $$Blue, -320, +85, +14495, -40 )
```

Grid Placement

Grids can be positioned at any location within a graphics window. They can nest within each other, but should generally not overlap.

Grids obscure each other**.** Grids are analogous to opaque objects.

Grid Advantages

Grids simplify graphics applications considerably. Programs can compute in natural units, create grids of any size, position them anywhere, and define their dimensions in any convenient units. Drawing in grids is then scaled and positioned automatically, and drawn with the current attributes of the grid.

Grid Attributes

Each grid has a number of *attributes* or *properties* that determine its appearance and behavior. The value of a particular property is called its *setting*. For example, **backgroundColor** is a property, while **$$Blue** is a setting it might have. *GraphicsDesigner* has functions to get and set all its grid attributes.

grid

grid is the grid number **XgrCreateGrid()** assigns a grid when it is created. **grid** is passed to all *GraphicsDesigner* functions that perform an operation associated with a grid.

window

window is the window number of the window that contains **grid**.

parent

parent is the grid number of another grid that contains and may take certain responsibilities for the operation of **grid**.

gridType

gridType is provided by *GraphicsDesigner* to make it easy for programs to groups similar grids into a single catagory. *GraphicsDesigner* assigns a new **gridType** each time **XgrRegisterGridType()** is called with a new grid type name.

gridType = 0 means the grid is a simple coordinate grid. Mouse events are never directed to coordinate grids, but to the next larger enabled grid that contains it that is not a coordinate grid.

gridType = 1 means the grid is an image grid. Drawing can be directed at image grids, but the image does not appear on the display, it is simply held in memory. On the other hand, the image can be transferred quickly to the display by *GraphicsDesigner* functions, and image grids can be attached to other grids to buffer them.

gridFunction

gridFunction is the function address called by **XgrSendMessage()** when it is called with a valid window or grid number and a valid message number.

bufferGrid

bufferGrid is the image grid that will buffer **grid**. The **gridType** of the grid assigned to **bufferGrid** must be **1**, which is the **gridType** reserved for image grids.

bufferGrid is initialized to **0** when a grid is created, which disables buffering and thereby speeds drawing operations.

font, fontName, fontSize, fontWeight, fontItalic,fontAngle

font is a font number created by **XgrCreateFont()** to refer to a specific font created to represent specified **fontName, fontSize, fontWeight, fontItalic, fontAngle**. **font** is passed to all *GraphicsDesigner* functions that require a font argument.

fontName is the typeface name of the font, **fontSize** is ten times the point size, **fontWeight** specifies the boldness of a font between 0 and 1000, **fontItalic** specifes the degree of italic tilt of a font between 0 and 1000, and **fontAngle** is the angle in 1/10th degrees the characters are rotated from normal horizontal orientation.

x,y,width,height

x,y,width,height are the position of **grid** within its parent grid, and its size in pixels. If a grid has no parent grid, **x,y** is its position within the window that contains it.

x1,y1:x2,y2

x1,y1:x2,y2 are the **upper-left:lower-right** corners of **grid** in local coordinates. **x1,y1** are fixed at **0,0** in local coordinates, while **x2,y2** are always **width-1,height-1**.

x1,y1:x2,y2 are established when **grid** is created by **XgrCreateGrid()** or resized by **XgrSetGridPositionAndSize()**.

x1Grid,y1Grid:x2Grid,y2Grid

x1Grid,y1Grid:x2Grid,y2Grid are the upper-left:lower-right corners of **grid** in grid coordinates. When a grid is created, its **x1Grid,y1Grid:x2Grid,y2Grid** is set to **0,0:width-1,height-1**.

No matter what values **x1Grid,y1Grid:x2Grid,y2Grid** contain, **x1Grid,y1Grid** are the grid coordinates of the grids upper-left corner and **x2Grid,y2Grid** are the grid coordinates of the grids lower-right corner.

Sometimes it is convenient to reverse x-axis and/or y-axis direction. **x2Grid < x1Grid** makes x-axis grid coordinates increase toward the left, while **y2Grid < y1Grid** makes y-axis grid coordinates increase toward the top.

x1#,y1#:x2#,y2#

x1#,y1#:x2#,y2# are the upper-left:lower-right corners of **grid** in scaled coordinates. When a grid is created, **x1#,y1#:x2#,y2#** are set to **0,0:x2,x2**, same as the local coordinates.

No matter what values **x1#,y1#:x2#,y2#** contain, **x1#,y1#** are the scaled coordinates of the grids upper-left corner and **x2#,y2#** are the scaled coordinates of the grids lower-right corner.

Sometimes it is convenient to reverse x-axis and/or y-axis direction. **x2# < x1#** makes x-axis scaled coordinates increase toward the left, while **y2# < y1#** makes y-axis scaled coordinates increase toward the top.

drawpoint, drawpointGrid, drawpointScaled

drawpoint, drawpointGrid, drawpointScaled are the current drawing coordinates for local, grid, and scaled coordinates, which means the final point specified in the previous draw or move function. For example, drawpoints are left at the endpoint of lines drawn line drawing functions, and at the position specified in move functions.

drawpoint, drawpointGrid, drawpointScaled are independent, one for each coordinate system.

backgroundColor

When a grid is cleared, it is filled with its **backgroundColor** unless specified otherwise in the function. When text is drawn with **XgrDrawTextFill()**, backgroundColor specifies the background color the text rectangle is cleared to before the text is drawn.

drawingColor

Points, lines, and text are drawn in the **drawingColor** unless specified otherwise in the function.

lowlightColor, highlightColor

When three dimensional shading effects are drawn at the border of grids, downslopes and upslopes are generally drawn in the **lowlightColor** and **highlightColor** respectively.

dullColor

When all or part of a grid needs to be deemphasized, part of it is usually drawn in its **dullColor** to make it less obvious and often to indicate some feature is temporarily disabled. For example, text that does not currently apply is often drawn in **dullColor**.

GraphicsDesigner does not draw deselect effects automatically. It only holds the **dullColor** setting so programs have a convenient place to keep all the important colors for each grid.

accentColor

When all or part of a grid needs to be emphasized, part of it is usually drawn in its **accentColor** to attract attention. For example, selected text items are often drawn in the **accentColor**.

GraphicsDesigner does not draw accent effects automatically. It only holds the **accentColor** setting so programs have a convenient place to keep the important colors for each grid.

lowtextColor, hightextColor

When text is drawn with three dimensional shading, the downslopes and upslopes of characters are generally drawn in the **lowtextColor** and **hightextColor** respectively.

GraphicsDesigner does not draw three dimensional text. It only holds **lowtextColor** and **hightextColor** so programs have a convenient place to keep the important colors for each grid.

Create Grid

XgrCreateGrid (@grid, gridType, x, y, w, h, window, parent, gridFunction)

creates a **grid** in **window**, positions its **upper-left** corner at **x,y** in the local coordinates of its parent, sets its local coordinates to **0,0:w-1,h-1**, and initializes its attributes to default values.

When **grid** is passed to graphics functions, operations are performed with the grid attributes.

Get and Set Grid Attributes

Functions **XgrGet*AttributeName*()** and **XgrSet*AttributeName*()** get and set the named grid attribute. These functions perform the action described by their name and arguments. For example:

```
  XgrGetDrawingRGB ( exhaustManifold, @red, @green, @blue )
  red=red+deltaRed : green=green+deltaGreen : blue=blue+deltaBlue
  XgrSetDrawingRGB ( exhaustManifold, red, green, blue )
'
  XgrGetGridBoxGrid ( grid, x1Grid, y1Grid, x2Grid, y2Grid )
  XgrSetBackgroundColor ( grid, $$DarkBlue )
```

Destroy Grid

XgrDestroyGrid() discards grids when they are no longer needed. It is important to destroy grids when they are not longer needed, since since every grid consumes about 1KB of system memory. **XgrDestroyGrid()** frees grid numbers for reuse.

XgrDestroyWindow() destroys all the grids in the window and removes the window from the display.

```
  XgrDestroyGrid ( grid )       ' destroy grid
  XgrDestroyWindow ( window )   ' destroy window and all grids in window
```

Image Grids

Image grids, or ***images***, are like regular grids except they're invisible. Graphics operations directed at them are stored in a memory image, not displayed. Image grids can be saved on disk, loaded from disk, copied into other image grids, and copied into regular grids.

Image grids are created by **`XgrCreateGrid()`**, when **`gridType=1`**. All graphics operations that apply to regular grids can be directed at image grids too.

```
XgrCreateGrid ( @grid,0,x,y,w,h,win,0,0 )              ' create grid
XgrCreateGrid ( @image0,1,x,y,w,h,win,0,0 )            '   and image0
XgrCreateGrid ( @image1,1,x,y,w,h,win,0,0 )            '   and image1
XgrLoadImage ( yourFace$, @image[] )                   ' image[]=face
XgrSetImage ( image0, @image[] )                       ' image0=face
XgrDrawImage ( image1,image0,startX,startY,endX,endY ) ' image1=face
XgrDrawImage ( grid,image0,startX,startY,endX,endY )   ' grid=face
XgrDestroyGrid ( image0 )                              ' trash image0
```

Buffering

When one window is moved over another, contents of the covered window are lost. When the top window is removed, the obscured area of the covered window is exposed, revealing an invalid image. Under such circumstances, programs must redraw their grids to update their windows. But some programs may find it difficult, time consuming, or even impossible to reconstruct the drawing area. Image grids provide an easy way around this problem.

Image grids can be attached to regular displayable grids. Graphics operations directed at the grid are also performed on the image. This has the effect of buffering the displayable grid.

```
XgrSetGridBuffer ( grid, image0 )        ' buffer grid with image0
XgrDrawLine ( grid,$$Blue,x1,y1,x2,y2 ) ' line in grid and image0
XgrDrawCircle ( image0, $$Red, radius ) ' draw circle on image0 only
XgrSetGridBuffer ( grid, 0 )             ' stop buffering grid
XgrClearGrid ( grid, $$Black )           ' clear grid to black
XgrRefreshGrid ( grid )                  ' line and circle on grid
XgrDestroyGrid ( image0 )                ' release image0 memory
```

When a grid needs to be redrawn, **`XgrRefreshGrid()`** will do it quickly and efficiently. Buffering displayable grids with image grids is convenient, but involves two costs. First, graphics operations directed at buffered grids are slowed significantly, since they are performed twice, once to the displayable grid and once to the image grid. Second, image grids consume memory. A 64x64 pixel image consumes between 2KB and 32KB, depending on the image type, while a 512x512 pixel image consumes 128KB to 2MB.

Buffering is performed automatically for grids that have a valid image grid attached. When image grids are created, their pixels are cleared to zero (black).

Color

Colors

Inside *GraphicsDesigner*, background and drawing colors are held as four **USHORT** values, one each for **red,green,blue** intensity **RGB**, plus one for a standard **color** number.

Programs can specify colors as:

- **Separate 16-bit values for (red,green,blue)**
- **32-bit value containing 8-bit values for (red,green,blue,color)**
- **Standard colorNumber between 0 and 124**

RGB Colors

16-bit per color **RGB** easily represents every displayable and printable color. Over ***48 trillion*** different colors can be specified in this **RGB** format. Human beings can distinguish only about 48 *million* colors.

Colors that are not red, green, or blue are created by combinations of red, green, blue intensities. The following table shows how a number of colors are often synthesized (values in hex):

Color	R-16	G-16	B-16	R-8	G-8	B-8	color#
$$Black	0000	0000	0000	00	00	00	00
$$DarkGrey	4000	4000	4000	40	40	40	1F
$$MediumGrey	8000	8000	8000	80	80	80	3E
$$LightGrey	C000	C000	C000	C0	C0	C0	5D
$$White	FFFF	FFFF	FFFF	FF	FF	FF	7C
$$DarkRed	4000	0000	0000	40	00	00	19
$$MediumRed	8000	0000	0000	80	00	00	32
$$BrightRed	C000	0000	0000	C0	00	00	4B
$$LightRed	FFFF	0000	0000	FF	00	00	64
$$LightGreen	0000	FFFF	0000	00	FF	00	14
$$LightBlue	0000	0000	FFFF	00	00	FF	04
$$LightCyan	0000	FFFF	FFFF	00	FF	FF	18
$$LightYellow	FFFF	FFFF	0000	FF	FF	00	78
$$LightMagenta	FFFF	0000	FFFF	FF	00	FF	3C
$$MediumAqua	4000	C000	8000	40	C0	80	2A
$$DarkBrown	4000	4000	0000	40	40	00	1E

The number of colors a computer can display at one time varies considerably from system to system. Some represent each primary color with an 8-bit intensity value, for 16 million simultaneously displayable colors. Others display only 256 different colors at one time, though each color can be any of these 16 million colors.

The colors actually displayed by *GraphicsDesigner* are as close to the specified colors as possible, given the state of computer system. No matter what system applications run on, they function identically, and graphics will appear as similar to the original results as practical.

Color Functions

Functions that draw into grids *do not* change their color attributes. Only functions that explicitly set colors alter color attributes. A **color** argument of **-1** means draw with the previously set colors.

The following functions convert colors between color formats, and get and set colors for a grid :

```
XgrConvertColorToRGB (color, @red, @green, @blue )
XgrConvertRGBToColor (red, green, blue, @color )
XgrGetBackgroundColor (grid, @color )
XgrGetBackgroundRGB (grid, @red, @green, @blue )
XgrGetDefaultColors (@back, @draw, @lo, @hi, @dull, @acc, @l, @h )
XgrGetDrawingColor (grid, @color )
XgrGetDrawingRGB (grid, @red, @green, @blue )
XgrGetGridColors (grid, @back, @draw, @lo, @hi, @dull, @acc, @l, @h )
XgrSetBackgroundColor (grid, color )
XgrSetBackgroundRGB (grid, red, green, blue )
XgrSetDefaultColors (back, draw, lo, hi, dull, acc, l, h )
XgrSetDrawingColor (grid, color )
XgrSetDrawingRGB (grid, red, green, blue )
XgrSetGridColors (grid, back, draw, lo, hi, dull, acc, l, h )
```

color

Graphics operations take an abbreviated **color** argument containing four 8-bit fields, one each for **red,green,blue** intensity, plus a standard **colorNumber** in the low byte. This argument is converted into internal **RGB** form as follows:

```
color = rgbc : 24-31 = r : 16-23 = g : 8-15 = b : 0-7 = colorNumber
color = 0              RGB = (0, 0, 0) = black
color = -1             RGB = current background or drawing color
colorNumber = 0        RGB = red << 24 : green << 16 : blue << 8
colorNumber < 124      RGB = from standard color[] array
colorNumber >= 124     RGB = reserved
```

The standard **colorNumber** is more than adequate for most applications. When accurate, high-quality color is needed, 8-bit per **RGB** is almost always sufficient. For the very highest quality image rendering, 8-bit per **RGB** is not quite sufficient, however.

16-bit per **RGB** is more than sufficient for the most demanding applications. 16-bit per **RGB** colors cannot be passed to drawing functions however. They must be set in advance by **XgrSetBackgroundRGB()** and/or **XgrSetDrawingRGB()**.

A **-1** color argument tells drawing functions to clear or draw in the previously set background or drawing color. A **-1** color argument tells color setting functions to leave the current value unchanged.

Functions that draw into grids *do not* change their color attributes. Only functions that explicitly set colors alter color attributes.

Standard Colors

An internal **standardColor[]** array contains a 5-intensity per color array of equally spaced colors created as follows:

```
DIM color[255]
DIM intensity[4]
intensity[0] = 0x0000   ' OFF
intensity[1] = 0x4000   ' 1/4 intensity
intensity[2] = 0x8000   ' 1/2 intensity
intensity[3] = 0xC000   ' 3/4 intensity
intensity[4] = 0xFFFF   ' full intensity
colorNumber = 0
FOR r = 0 TO 4
  red = intensity[r]
  FOR g = 0 TO 4
    green = intensity[g]
    FOR b = 0 TO 4
      blue = intensity[b]
      color[colorNumber].r = red
      color[colorNumber].g = green
      color[colorNumber].b = blue
      color[colorNumber].x = colorNumber
      INC colorNumber
    NEXT
  NEXT
NEXT

color = 25 * red + 5 * green + blue
```

red,green,blue are intensities **0 to 4**.

Human beings do not perceive color intensities linearly. It is much harder to distinguish similar colors from each other when they are dark or bright, as opposed to intermediate. Some computers and system software compensate by adjusting colors so equally spaced intensities *appear* equally spaced. If they don't, *GraphicsDesigner* performs color compensation, but only if it knows this kind of compensation hasn't already been performed, and the correction can be made without replacing solid colors with colors simulated by dithering. Color compensation has no effect on program values.

Messages

Messages

When certain kinds of events take place in graphics windows, *GraphicsDesigner* creates *messages* to describe them.

For example, when a keyboard key is pressed, *GraphicsDesigner* creates a **`WindowKeyDown`** message. When the mouse moves or a button is depressed, a **`MouseMove`** or **`MouseDown`** message is created. When a user clicks on a window, a **`WindowSelected`** message is created.

In general, whenever the state of the keyboard, mouse, or a window changes, *GraphicsDesigner* creates a message that describes it.

Only events in graphics windows are detected by *GraphicsDesigner*, so only events in graphics windows cause the creation of messages.

Window Messages and Grid Messages

Any particular message name, like **`WindowKeyDown`** or **`MouseDown`**, refers to either windows or grids, *never both*.

Window messages refer to windows.

Grid messages refer to grids.

Window message names *must* start with **`"Window"`**.
Grid message names *must not* start with **`"Window"`**.

Message Anatomy

Every *GraphicsDesigner* message contains **8 XLONG** values:

```
( window, message, v0, v1, v2, v3, r0, r1 )
 ... or ...
( grid, message, v0, v1, v2, v3, r0, r1 )
 ... or ...
( wingrid, message, v0, v1, v2, v3, r0, r1 )
```

window, grid, wingrid

window contains the window number the message refers to.

grid contains the grid number the message refers to.

wingrid contains the window or grid number a message refers to. **wingrid** is the name given to arguments that contain a window number in some contexts, and a grid number in others.

message *aka message number*

message contains a message number. Message numbers are initially based on strings like **"MouseDown"**, but messages contain message numbers so programs must get the message number for every message they use when they start up and assign them to similar named variables like **#MouseDown**.

The message number for any message name string is returned by two functions, the only difference being that **XgrRegisterMessage()** creates a new message number for message names that don't already exist, while **XgrMessageNameToNumber()** returns zero, which is an invalid message number.

```
XgrRegisterMessage ( messsage$, @message )
XgrMessageNameToNumber ( message$, @message )
```

v0,v1,v2,v3,r0,r1

v0,v1,v2,v3,r0,r1 contain message arguments whose meanings depend on **message**. For example, in mouse messages **v0,v1,v2,v3,r0,r1** contain **x,y,state,time,0,0**.

In some messages, one or more arguments contains no defined value. *GraphicsDesigner* and *GuiDesigner* always fill these arguments with zero when calling other programs, and expect to receive zeros in these arguments from other programs.

Message Queue

GraphicsDesigner combines these arguments into a complete message and places it in a *message queue.* Since it is built into *GraphicsDesigner*, the message queue is accessible by all programs.

GraphicsDesigner stores messages in the message queue until a program is ready to process them. Programs receive messages in the order they were put in the queue. First come, first served. Or in computer lingo, first-in, first-out, or FIFO.

Programs can add their own messages to the queue too. The message queue is a general purpose mechanism for one part of a program to communicate with another, even if the program creates no windows or graphics at all.

Process Message

Programs can call **`XgrPeekMessage()`** to extract a message from the queue, examine its arguments, and possibly call other functions and/or take actions in response.

```
XgrPeekMessage ( @grid, @message, @v0, @v1, @v2, @v3, @r0, @r1 )
XgrDeleteMessages ( 1 )
[ ... examine and take action based on grid,message,v0,v1,v2,v3,r0,r1 ]
```

Alternatively, programs can have *GraphicsDesigner* call the appropriate functions to process **`n`** messages by calling:

```
XgrProcessMessages (n)
```

Window Function

Windows can be assigned a *window function* when they are created by **XgrCreateWindow()**, or later by **XgrSetWindowFunction()**.

XgrProcessMessages() calls the window function the message refers to directly or indirectly...

For window messages, **XgrProcessMessages()** calls the window function assigned to the window specified by the **window** argument.

For grid messages, **XgrProcessMessages()** calls the window function assigned to the window that contains the grid specified by the **grid** argument.

XgrProcessMessages() calls window functions as follows:

```
@func ( wingrid, message, v0, v1, v2, v3, 0, wingrid )
```

XgrProcessMessages() calls functions that expect eight **XLONG** arguments.

Window functions are usually declared and defined as follows:

```
DECLARE FUNCTION  Name ( wingrid, message, v0, v1, v2, v3, r0, ANY )
FUNCTION  Name ( wingrid, message, v0, v1, v2, v3, r0, (r1, r1$) )
```

Grid Functions

When window functions receive a *window message*, they can ignore it or take some action. When finished, they return to the calling function, which is generally **XgrProcessMessages()**.

When window functions receive a *grid message*, they can ignore it, take some action, or pass it on to the *grid function* whose address was **func** in the **XgrCreateGrid()** call that created the grid. The easiest way to call grid functions is to pass the arguments to **XgrSendMessage()** and let **XgrSendMessage()** look up the grid function assigned to **grid** and call it, as in:

```
XgrSendMessage ( wingrid, message, v0, v1, v2, v3, 0, wingrid )
```

XgrSendMessage() calls only 8 argument functions, so all grid functions must take eight arguments. The first seven arguments are always **XLONG**, while the type of the last can be **XLONG**, **STRING**, a composite type, or an array of any valid type.

Grid functions are usually declared and defined as follows:

```
DECLARE FUNCTION  Name ( grid, message, v0, v1, v2, v3, r0, ANY )
FUNCTION  Name ( grid, message, v0, v1, v2, v3, r0, (r1, r1$) )
```

Sending Messages

To *send a message* means to call a window function, a grid function, or any other function designed to take window and/or grid messages.

To *send a message to a window* means to call the window function associated with the window argument in the message.

To *send a message to a grid* means to call the grid function associated with the grid argument in the message.

To think of "sending a message to a grid" may seem strange at first, since sending messages is not part of the programming language. But "send message" is a lot more compact than "look up the window or grid function assigned to the window or grid argument in the message and call the function, passing the message arguments".

When you write *GuiDesigner* programs, you'll find the notion of sending a message is practically indespensible.

Program Wide Messages

Sending messages to windows and grids simplifies the structure of a program by directing messages to only that part of a program that is designed to handle it. But certain events are important or relevant to all or most parts of a program.

For example, many programs display a set of function keys that are supposed to perform certain functions whenever they are pressed, regardless of which window is selected. When processed by **`XgrProcessMessages()`**, however, keyboard messages for function keys are sent to the same place as any keyboard message - to the window function of the selected window. Since programs can have several window functions, every window function would have to be prepared to check for function keys and take appropriate action.

To make it easier for programs to process messages that are relevant to several parts of a program, *GraphicsDesigner* lets each program register one *CEO function*.

CEO Function

A program can register only one *CEO function*, but **`XgrProcessMessages()`**, sends every message it processes to that function, no matter what the message contains. Furthermore, **`XgrProcessMessages()`** sends each message to the **`CEO`** function before any other, and the **`CEO`** can cancel the message to prevent its propogation to other functions.

Programs call **`XgrSetCEO(func)`** to set the address of the **`CEO`** function to **`func`**. An address of **`0`** means no **`CEO`** function is active. Only one **`CEO`** function can be active at a time, so each time **`XgrSetCEO(func)`** is called, the previous **`CEO`** function is replaced by **`func`**.

When messages are processed by **`XgrProcessMessages()`**, they are sent to the **`CEO`** function, then to the window function appropriate to the window/grid and message. This sequence is initiated when programs call **`XgrProcessMessages()`**.

`XgrProcessMessages()` checks to see if a **`CEO`** function exists. If it does, it calls the **`CEO`** function, passing it the message arguments, plus **`0`** in **`r0`** and a duplicate of the window or grid argument in **`r1`**. The **`CEO`** can examine the message and take whatever action it needs to perform its function.

The **`CEO`** function can return **`-1`** in **`r0`** to cancel the message. When **`XgrProcessMessages()`** finds the CEO function returned **`-1`** in **`r0`**, it cancels the message and returns without calling any window functions it otherwise would have called to process the message.

Messages NOT

Simple graphics programs don't need messages or the message queue, and can simpy ignore them. Only programs that need to detect keyboard and mouse events in graphics windows need messages.

Messages Simple

Programs that need to monitor keyboard and/or mouse activities in graphics windows have to recognize keyboard and/or mouse messages, but can discard others.

`XgrPeekMessage()` and **`XgrDeleteMessages()`** are all they need to extract a message from the queue, take whatever action is appropriate, then remove the message from the queue so the next one becomes accessible. In this method, only your own program calls functions in your program.

Messages Advanced

When a program controls several windows, it's sometimes appropriate to write separate functions for windows that have special functionality that's awkward to handle in a single window function.

Whenever a program is ready to process a message, it can call **`XgrProcessMessages()`** and let it call the window function assigned to the window.

Messages Sophisticated

Sophisticated programs, like those with graphical user interfaces often send, receive, and process a wide variety of messages. Often they have a wide variety of special purpose grids, supported by one grid function per grid type.

The *GuiDesigner* programmer guide contains additional discussion of messages and the message queue, since message passing is a central part of its architecture. Furthermore, *GuiDesigner* contains additional message functions that work together to support sophisticated message passing for any program, whether they perform graphics or not.

GraphicsDesigner Messages

The messages in the following table are put in the message queue by *GraphicsDesigner* in response to system events. **`XgrProcessMessages()`** sends these messages to window functions when it processes them, unless no window function is defined for the associated window.

In the following table, window and grid messages are distinguished by a **`w`** or **`g`** before the message name. More detailed descriptions of these messages follows the "Message Functions" descriptions at the end of the next section.

g	MouseDown	A mouse button was depressed.
g	MouseDrag	The mouse moved while one or more buttons was down.
g	MouseEnter	The mouse cursor moved into a grid.
g	MouseExit	The mouse cursor moved out of a grid.
g	MouseMove	The mouse moved while no buttons were down.
g	MouseUp	A mouse button was released.
g	RedrawGrid	Redraw a grid to update its contents.
g	TimeOut	A grid timer started with XgrSetGridTimer() counted down to zero.
w	WindowDeselected	A window was deselected and no longer has keyboard focus.
w	WindowDestroyed	A window was destroyed.
w	WindowDisplayed	A window was displayed.
w	WindowHidden	A window was hidden or minimized.
w	WindowKeyDown	A keyboard key was depressed.
w	WindowKeyUp	A keyboard key was released.
w	WindowRedraw	A window needs to be redrawn partially or completely.
w	WindowResized	A window was resized.
w	WindowSelected	A window was selected and now has keyboard focus.

Keyboard Messages

Keyboard messages contain the **window** number of the window that was selected when the keyboard event was detected.

v0,v1,v2,v3,r0,r1 contain **x,y,state,time,0,grid**.

x,y

x,y contain the position of the mouse cursor in the local coordinates of the specified **grid** at the time the keyboard event was detected. If **x** or **y** is negative, the mouse cursor was outside the specified grid, or the mouse coordinates were unavailable.

state

state contains the state of the keyboard when the keyboard event was detected, and reflects the new state of the keyboard.

```
Bit 00 - 15 :  Character code of some kind (see bits 20-21)
Bit 16 - 23 :  Keyboard "mode" keys (16=Shift, 17=Control, 18=Alt)
Bit 24 - 31 :  Virtual Key Code

Bit 16 = 1  : Shift key was down when the keyboard event occured.
Bit 17 = 1  : Control key was down when the keyboard event occured.
Bit 18 = 1  : Alt key was down when the keyboard event occured.
Bit 19      : Reserved (right Alt key down?)
Bit 20 - 21 : Type of character code in Bit 00 - 15  (see below)
Bit 20 - 21 : 0 = Bit 00 - 15 = Virtual Key Code (8-bits)
            : 1 = Bit 00 - 15 = ASCII character (8-bits)
            : 2 = Bit 00 - 15 = WIDE character (16-bits)
            : values 3 to 7 are reserved
Bit 22      : Reserved (right Shift key down?)
Bit 23      : Reserved (right Control key down?)
```

time

time contains the system millisecond time that the keyboard event was detected. **time** is not related to time of day. It is simply a free running millisecond timer that computer systems usually initialize to zero when they are started.

Keyboard Message Examples

Key	24-31	20-21	18	17	16	0-15	"."	mode key states, key event
Down	97	1	0	0	0	97	a	None down, "a" down
Up	97	0	0	0	0	97	-	None down, "a" up
Down	65	1	0	0	1	65	A	Shift down, "a" down
Up	65	0	0	0	1	65	-	Shift down, "a" up
Down	65	0	0	1	0	65	^A	Ctl down, "a" down
Down	65	0	0	1	1	65	?	Ctl+Shift down, "a" down
Down	65	0	1	0	0	65	?	Alt down, "a" down
Down	65	0	1	0	1	65	?	Alt+Shift down, "a" down
Down	65	0	1	1	0	65	?	Alt+Ctl down, "a" down
Down	65	0	1	1	1	65	?	Alt+Ctl+Shift down, "a" down
Up	65	0	1	1	1	65	?	Alt+Ctl+Shift down, "a" up
Down	39	0	0	1	0	39	Left	Ctl down, LeftArrow press

WindowKeyDown vs WindowKeyUp

Most programs respond to **`WindowKeyDown`** messages, but not **`WindowKeyUp`** messages, because that is sufficient to react to all keystrokes.

`WindowKeyDown` messages are created at a rate of about 20 per second if a key is held down for more than about .5 seconds, followed by a single **`WindowKeyUp`** when the key is released.

Virtual Key Codes

8	0x08	KeyBackspace
9	0x09	KeyTab
12	0x0C	KeyClear
13	0x0D	KeyEnter
16	0x10	KeyShift
17	0x11	KeyControl
18	0x12	KeyAlt
19	0x13	KeyPause
20	0x14	KeyCapLock
27	0x1B	KeyEscape
32	0x20	KeySpace
33	0x21	KeyPageUp
34	0x22	KeyPageDown
35	0x23	KeyEnd
36	0x24	KeyHome
37	0x25	KeyLeftArrow
38	0x26	KeyUpArrow
39	0x27	KeyRightArrow
40	0x28	KeyDownArrow
44	0x2C	KeyPrintScreen
45	0x2D	KeyInsert
46	0x2E	KeyDelete
47	0x2F	KeyHelp
48	0x30	Key0
49	0x31	Key1
50	0x32	Key2
51	0x33	Key3
52	0x34	Key4
53	0x35	Key5
54	0x36	Key6
55	0x37	Key7
56	0x38	Key8
57	0x39	Key9
65	0x41	KeyA
66	0x42	KeyB
67	0x43	KeyC
68	0x44	KeyD
69	0x45	KeyE
70	0x46	KeyF
71	0x47	KeyG
72	0x48	KeyH
73	0x49	KeyI
74	0x4A	KeyJ
75	0x4B	KeyK
76	0x4C	KeyL
77	0x4D	KeyM
78	0x4E	KeyN
79	0x4F	KeyO
80	0x50	KeyP
81	0x51	KeyQ
82	0x52	KeyR
83	0x53	KeyS
84	0x54	KeyT
85	0x55	KeyU
86	0x56	KeyV
87	0x57	KeyW
88	0x58	KeyX
89	0x59	KeyY
90	0x5A	KeyZ
96	0x60	KeyPad0
97	0x61	KeyPad1
98	0x62	KeyPad2
99	0x63	KeyPad3
100	0x64	KeyPad4
101	0x65	KeyPad5
102	0x66	KeyPad6
103	0x67	KeyPad7
104	0x68	KeyPad8
105	0x69	KeyPad9
106	0x6A	KeyPadMultiply
107	0x6B	KeyPadAdd
108	0x6C	_
109	0x6D	KeyPadSubtract
110	0x6E	KeyPadDecimalPoint
111	0x6F	KeyPadDivide
112	0x70	KeyF1
113	0x71	KeyF2
114	0x72	KeyF3
115	0x73	KeyF4
116	0x74	KeyF5
117	0x75	KeyF6
118	0x76	KeyF7
119	0x77	KeyF8
120	0x78	KeyF9
121	0x79	KeyF10
122	0x7A	KeyF11
123	0x7B	KeyF12
124	0x7C	KeyF13
125	0x7D	KeyF14
126	0x7E	KeyF15
127	0x7F	KeyF16
144	0x90	KeyNumLock

Mouse Messages

Mouse messages contain the grid number of the **grid** the message is prepared for and has mouse focus. When no mouse buttons are down, the grid that contains the mouse cursor is also the mouse focus grid. When a mouse button is depressed when none are currently down, the grid that contains the mouse cursor automatically *grabs* mouse focus and holds it until all mouse buttons are released, at which time mouse focus goes to the grid that contains the mouse cursor. Mouse messages are normally routed to the grid with mouse focus, but **MouseExit** and **MouseEnter** messages may be generated for other grids as the mouse cursor moves from grid to grid while mouse focus is grabbed.

v0,v1,v2,v3,r0,r1 contain **x,y,state,time,0,grid**.

x,y

x,y contain the position of the mouse cursor in the local coordinates of **focusGrid** at the time the mouse event was detected.

x,y may indicate a mouse cursor position outside **grid** and/or **focusGrid** if mouse focus has been grabbed by **focusGrid**.

state

```
Bit 00 - 03 :  Button # causing event (MouseDown and MouseUp only )
Bit 04 - 06 :  # of clicks (MouseDown only)
Bit      07 :  1 if grid has mouse focus
Bit 08 - 15 :  Reserved
Bit 16 - 23 :  Keyboard "mode" keys  (16=Shift, 17=Control, 18=Alt)
Bit 24 - 31 :  Up/Down image of up to 8 mouse buttons (1 = down)

Bit 00 - 03 :  Button #:  None=0 : Left=1 : Center=2 : Right=3...
Bit 16 = 1  :  Shift key is down
Bit 17 = 1  :  Control key is down
Bit 18 = 1  :  Alt key is down
Bit 24 = 1  :  Left button is down
Bit 25 = 1  :  Center button is down
Bit 26 = 1  :  Right button is down
Bit 27 - 31 :  Other buttons down (assignments not guaranteed)
```

time

time contains the system millisecond time that the keyboard event was detected. **time** is not related to time of day. It is simply a free running millisecond timer that computer systems usually initialize to zero when they are started.

GraphicsDesigner Functions

Function Categories

The tables on the following pages list the *GraphicsDesigner* functions, grouped into the following catagories for easy access.

Miscellaneous	**Display information**
Color Functions	**Get and Set Background and Drawing Colors**
Window Functions	**Create, Destroy, Position, Size, Iconify**
Grid Functions	**Create, Destroy, GetInfo, SetInfo, GridBox**
Drawing Functions	**Arc, Box, Circle, Line, Point**
Image Functions	**Load, Save, Copy, RefreshGrid**
Focus Functions	**Keyboard and Mouse Focus**
Message Functions	**Add, Delete, Get, Jam, Peek, Process, Send**
Messages	**KeyDown, KeyUp, MouseDown, MouseDrag...**

Reference Pages

Following the function tables are reference pages that describe these functions in more detail, organized in the same order as the listings.

Arguments - Pass By Reference

Function arguments with `@` prefixes return a value in the argument. In some cases, arguments are passed to a function and back from the function in the same variable. `@` is prefixed to all arrays because arrays are always passed by reference. String and array contents are not modified unless so stated.

Return Values

Most *GraphicsDesigner* functions do not return a value. Exceptions are noted in the following sections.

Runtime Errors

GraphicsDesigner functions call **`ERROR()`** to log errors, and may return a non-zero value to so indicate. Errors can be retrieved, and cleared or set by intrinsic function **`ERROR()`**.

GraphicsDesigner Function Quick Reference

The following summary of *GraphicsDesigner* functions are grouped by category.

Miscellaneous Functions

```
Xgr                          ( )
XgrCreateFont                ( @font, fontName$, fontSize, fontWeight, fontItalic, fontAngle )
XgrBorderNameToNumber        ( border$, @border )
XgrBorderNumberToName        ( border, @border$ )
XgrBorderNumberToWidth       ( border, @width )
XgrCursorNameToNumber        ( cursorName$, @cursorNumber )
XgrCursorNumberToName        ( cursorNumber, @cursorName$ )
XgrDestroyFont               ( font )
XgrGetClipboard              ( clipboard, @clipType, @text$, @image[] )
XgrGetCursor                 ( @cursor )
XgrGetDisplaySize            ( display$, @width, @height, @borderWidth, @titleHeight )
XgrGetFontInfo               ( font, @fontName$, @fontSize, @fontWeight, @fontItalic, fontAngle )
XgrGetFontMetrics            ( font, @maxCharWidth, @maxCharHeight, @ascent, @decent, @gap, @flags )
XgrGetFontNames              ( @count, @fontNames$[] )
XgrGetKeystateModify         ( state, @modify, @edit )
XgrGetTextImageSize          ( font, text$, @dx, @dy, @width, @height, @gap, @flags )
XgrIconNameToNumber          ( iconName$, @iconNumber )
XgrIconNumberToName          ( iconNumber, @iconName$ )
XgrRegisterCursor            ( cursorName$, @cursor )
XgrRegisterIcon              ( iconName$, @icon )
XgrSetClipboard              ( clipboard, clipType, @text$, @image[] )
XgrSetCursor                 ( cursor, @oldCursor )
version$ = XgrVersion$       ( )
```

Color Functions

```
XgrConvertColorToRGB         ( color, @red, @green, @blue )
XgrConvertRGBToColor         ( red, green, blue, @color )
XgrGetBackgroundColor        ( grid, @color )
XgrGetBackgroundRGB          ( grid, @red, @green, @blue )
XgrGetDefaultColors          (@back, @draw, @lo, @hi, @acc, @dull )
XgrGetDrawingColor           ( grid, @color )
XgrGetDrawingRGB             ( grid, @red, @green, @blue )
XgrGetGridColors             ( grid, @back, @draw, @lo, @hi, @acc, @dull )
XgrSetBackgroundColor        ( grid, color )
XgrSetBackgroundRGB          ( grid, red, green, blue )
XgrSetDefaultColors          ( back, draw, lo, hi, acc, dull )
XgrSetDrawingColor           ( grid, color )
XgrSetDrawingRGB             ( grid, red, green, blue )
XgrSetGridColors             ( grid, back, draw, lo, hi, acc, dull )
```

Window Functions

```
XgrClearWindow               ( window, color )
XgrClearWindowAndImages      ( window, color )
XgrCreateWindow              (@window, winType, @xDisp, @yDisp, @width, @height, winFunc, display$ )
XgrDestroyWindow             ( window )
XgrDisplayWindow             ( window )
XgrGetWindowFunction         ( window, @func )
XgrGetWindowIcon             ( window, @icon )
XgrGetWindowPositionAndSize  ( window, @xDisp, @yDisp, @width, @height )
XgrGetWindowState            ( window, @state )
XgrGetWindowTitle            ( window, @title$ )
XgrHideWindow                ( window )
XgrMaximizeWindow            ( window )
XgrMinimizeWindow            ( window )
XgrRestoreWindow             ( window )
XgrSetWindowFunction         ( window, func )
XgrSetWindowIcon             ( window, @icon )
XgrSetWindowPositionAndSize  ( window, xDisp, yDisp, width, height )
XgrSetWindowState            ( window, state )
XgrSetWindowTitle            ( window, title$ )
XgrShowWindow                ( window )
```

```
XgrClearGrid                  ( grid, color )
XgrConvertDisplayToGrid       ( grid, xDisp, yDisp, @xGrid, @yGrid )
XgrConvertDisplayToLocal      ( grid, xDisp, yDisp, @x, @y )
XgrConvertDisplayToScaled     ( grid, xDisp, yDisp, @x#, @y# )
XgrConvertDisplayToWindow     ( grid, xDisp, yDisp, @xWin, @yWin )
XgrConvertGridToDisplay       ( grid, xGrid, yGrid, @xDisp, @yDisp )
XgrConvertGridToLocal         ( grid, xGrid, yGrid, @x, @y )
XgrConvertGridToScaled        ( grid, xGrid, yGrid, @x#, @y# )
XgrConvertGridToWindow        ( grid, xGrid, yGrid, @xWin, @yWin )
XgrConvertLocalToDisplay      ( grid, x, y, @xDisp, @yDisp )
XgrConvertLocalToGrid         ( grid, x, y, @xGrid, @yGrid )
XgrConvertLocalToScaled       ( grid, x, y, @x#, @y# )
XgrConvertLocalToWindow       ( grid, x, y, @xWin, @yWin )
XgrConvertScaledToDisplay     ( grid, x#, y#, @xDisp, @yDisp )
XgrConvertScaledToGrid        ( grid, x#, y#, @xGrid, @yGrid )
XgrConvertScaledToLocal       ( grid, x#, y#, @x, @y )
XgrConvertScaledToWindow      ( grid, x#, y#, @xWin, @yWin )
XgrConvertWindowToDisplay     ( grid, xWin, yWin, @xDisp, @yDisp )
XgrConvertWindowToGrid        ( grid, xWin, yWin, @xGrid, @yGrid )
XgrConvertWindowToLocal       ( grid, xWin, yWin, @x, @y )
XgrConvertWindowToScaled      ( grid, xWin, yWin, @x#, @y# )
XgrCreateGrid                 (@grid, gridType, xWin, yWin, width, height, window, parent, func )
XgrDestroyGrid                ( grid )
XgrGetGridBoxGrid             ( grid, @x1Grid, @y1Grid, @x2Grid, @y2Grid )
XgrGetGridBoxLocal            ( grid, @x1, @y1, @x2, @y2 )
XgrGetGridBoxScaled           ( grid, @x1#, @y1#, @x2#, @y2# )
XgrGetGridBoxWindow           ( grid, @x1Win, @y1Win, @x2Win, @y2Win )
XgrGetGridBuffer              ( grid, @bufferGrid )
XgrGetGridCharacterMapArray   ( grid, @map[] )
XgrGetGridCoords              ( grid, @xParent, @yParent, @x1Grid, @y1Grid, @x2Grid, @y2Grid )
XgrGetGridDrawingMode         ( grid, @drawingMode, @lineStyle, @lineWidth )
XgrGetGridFont                ( grid, @font )
XgrGetGridFunction            ( grid, @func )
XgrGetGridParent              ( grid, @parent )
XgrGetGridPositionAndSize     ( grid, @xWin, @yWin, @width, @height )
XgrGetGridState               ( grid, @state )
XgrGetGridType                ( grid, @gridType )
XgrGetGridWindow              ( grid, @window )
XgrGridTypeNameToNumber       ( gridType$, @gridType )
XgrGridTypeNumberToName       ( gridType, @gridType$ )
XgrRegisterGridType           ( gridType$, @gridType )
XgrSetGridBoxGrid             ( grid, x1Grid, y1Grid, x2Grid, y2Grid )
XgrSetGridBoxScaled           ( grid, x1#, y1#, x2#, y2# )
XgrSetGridBuffer              ( grid, bufferGrid )
XgrSetGridCharacterMapArray   ( grid, @map[] )
XgrSetGridDrawingMode         ( grid, drawingMode, lineStyle, lineWidth )
XgrSetGridFont                ( grid, font )
XgrSetGridFunction            ( grid, funcAddr )
XgrSetGridPositionAndSize     ( grid, xWin, yWin, width, height )
XgrSetGridState               ( grid, state )
XgrSetGridTimer               ( grid, msTimeInterval )
XgrSetGridType                ( grid, gridType )
```

```
XgrDrawArc                    ( grid, color, r, startAngle#, endAngle# )
XgrDrawArcGrid                ( grid, color, r, startAngle#, endAngle# )
XgrDrawArcScaled              ( grid, color, r#, startAngle#, endAngle# )
XgrDrawBorder                 ( grid, border, draw, low, high, x1, y1, x2, y2 )
XgrDrawBorderGrid             ( grid, border, draw, low, high, x1Grid, y1Grid, x2Grid, y2Grid )
XgrDrawBorderScaled           ( grid, border, draw, low, high, x1#, y1#, x2#, y2# )
XgrDrawBox                    ( grid, color, x1, y1, x2, y2 )
XgrDrawBoxGrid                ( grid, color, x1Grid, y1Grid, x2Grid, y2Grid )
XgrDrawBoxScaled              ( grid, color, x1#, y1#, x2#, y2# )
XgrDrawCircle                 ( grid, color, r )
XgrDrawCircleGrid             ( grid, color, r )
XgrDrawCircleScaled           ( grid, color, r# )
XgrDrawGridBorder             ( grid, border )
XgrDrawIcon                   ( grid, icon, x, y )
XgrDrawIconGrid               ( grid, icon, xGrid, yGrid )
XgrDrawIconScaled             ( grid, icon, x#, y# )
XgrDrawLine                   ( grid, color, x1, y1, x2, y2 )
XgrDrawLineGrid               ( grid, color, x1Grid, y1Grid, x2Grid, y2Grid )
XgrDrawLineScaled             ( grid, color, x1#, y1#, x2#, y2# )
XgrDrawLineTo                 ( grid, color, x, y )
XgrDrawLineToGrid             ( grid, color, xGrid, yGrid )
XgrDrawLineToScaled           ( grid, color, x#, y# )
XgrDrawLineToDelta            ( grid, color, dx, dy )
XgrDrawLineToDeltaGrid        ( grid, color, dxGrid, dyGrid )
XgrDrawLineToDeltaScaled      ( grid, color, dx#, dy# )
XgrDrawLines                  ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesGrid              ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesScaled            ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesTo                ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesToGrid            ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesToScaled          ( grid, color, first, count, ANY @lines[] )
XgrDrawPoint                  ( grid, color, x, y )
XgrDrawPointGrid              ( grid, color, xGrid, yGrid )
XgrDrawPointScaled            ( grid, color, x#, y# )
XgrDrawPoints                 ( grid, color, first, count, ANY @points[] )
XgrDrawPointsGrid             ( grid, color, first, count, ANY @points[] )
XgrDrawPointsScaled           ( grid, color, first, count, ANY @points[] )
XgrDrawText                   ( grid, color, text$ )
XgrDrawTextGrid               ( grid, color, text$ )
XgrDrawTextScaled             ( grid, color, text$ )
XgrDrawTextFill               ( grid, color, text$ )
XgrDrawTextFillGrid           ( grid, color, text$ )
XgrDrawTextFillScaled         ( grid, color, text$ )
XgrFillBox                    ( grid, color, x1, y1, x2, y2 )
XgrFillBoxGrid                ( grid, color, x1Grid, y1Grid, x2Grid, y2Grid )
XgrFillBoxScaled              ( grid, color, x1#, y1#, x2#, y2# )
XgrGetDrawpoint               ( grid, @x, @y )
XgrGetDrawpointGrid           ( grid, @xGrid, @yGrid )
XgrGetDrawpointScaled         ( grid, @x#, @y# )
XgrGrabPoint                  ( grid, x, y, @red, @green, @blue, @color )
XgrGrabPointGrid              ( grid, xGrid, yGrid, @red, @green, @blue, @color )
XgrGrabPointScaled            ( grid, x#, y#, @red, @green, @blue, @color )
XgrMoveDelta                  ( grid, dx, dy )
XgrMoveDeltaGrid              ( grid, dxGrid, dyGrid )
XgrMoveDeltaScaled            ( grid, dx#, dy# )
XgrMoveTo                     ( grid, x, y )
XgrMoveToGrid                 ( grid, xGrid, yGrid )
XgrMoveToScaled               ( grid, x#, y# )
XgrSetDrawpoint               ( grid, x, y )
XgrSetDrawpointGrid           ( grid, xGrid, yGrid )
XgrSetDrawpointScaled         ( grid, x#, y# )
```

Image Functions

```
XgrCopyImage                    ( grid, imageGrid )
XgrDrawImage                    ( grid, imageGrid, startX, startY, endX, endY )
XgrDrawImageExtend              ( grid, imageGrid, startX, startY, endX, endY )
XgrDrawImageExtendScaled        ( grid, imageGrid, startX, startY, endX, endY )
XgrDrawImageScaled              ( grid, imageGrid, startX, startY, endX, endY )
XgrGetImage                     ( imageGrid, @image[] )
XgrGetImageArrayInfo            (@image[], @bitsPerPixel, @width, @height )
XgrLoadImage                    ( fileName$, @image[] )
XgrRefreshGrid                  ( grid )
XgrSaveImage                    ( fileName$, @image[] )
XgrSetImage                     ( imageGrid, @image[] )
```

Focus Functions

```
XgrGetMouseInfo                 ( window, @grid, @xWin, @yWin, @state, @time )
XgrGetSelectedWindow            ( @window )
XgrSetSelectedWindow            ( window )
```

Message Functions

```
XgrAddMessage                   ( wingrid, message, v0, v1, v2, v3 )
XgrDeleteMessages               ( count )
XgrGetCEO                       (@func )
XgrGetMessages                  (@count, @messages[] )
XgrGetMessageType               ( message, @messageType )
XgrJamMessage                   ( wingrid, message, v0, v1, v2, v3 )
XgrMessageNameToNumber          ( message$, @message )
XgrMessageNames                 (@count, @messages$[] )
XgrMessageNumberToName          ( message, @message$ )
XgrMessagesPending              (@count )
XgrPeekMessage                  (@wingrid, @message, @v0, @v1, @v2, @v3 )
XgrProcessMessages              ( maxCount )
XgrRedrawWindow                 ( window, xWin, yWin, width, height )
XgrRegisterMessage              ( message$, @message )
XgrSendMessage                  ( wingrid, message, v0, v1, v2, v3, r0, r1 )
XgrSendMessageToWindow          ( wingrid, message, v0, v1, v2, v3, r0, r1 )
XgrSendStringMessage            ( wingrid, message$, v0, v1, v2, v3, r0, r1 )
XgrSendStringMessageToWindow    ( wingrid, message$, v0, v1, v2, v3, r0, r1 )
XgrSetCEO                       ( func )
```

Messages

```
MouseDown                       ( grid, MouseDown, x, y, state, time, 0, focusGrid )
MouseDrag                       ( grid, MouseDrag, x, y, state, time, 0, focusGrid )
MouseEnter                      ( grid, MouseEnter, x, y, state, time, 0, focusGrid )
MouseExit                       ( grid, MouseExit, x, y, state, time, 0, focusGrid )
MouseMove                       ( grid, MouseMove, x, y, state, time, 0, focusGrid )
MouseUp                         ( grid, MouseUp, x, y, state, time, 0, focusGrid )
RedrawGrid                      ( grid, RedrawGrid, x, y, width, height, 0, 0 )
TimeOut                         ( grid, TimeOut, 0, 0, 0, 0, 0, 0 )
WindowDeselected                ( window, WindowDeselected, 0, 0, 0, 0, 0, 0 )
WindowDestroyed                 ( window, WindowDestroyed, 0, 0, 0, 0, 0, 0 )
WindowDisplayed                 ( window, WindowDisplayed, 0, 0, 0, 0, 0, 0 )
WindowHidden                    ( window, WindowHidden, 0, 0, 0, 0, 0, 0 )
WindowKeyDown                   ( window, WindowKeyDown, x, y, state, time, 0, 0 )
WindowKeyUp                     ( window, WindowKeyUp, x, y, state, time, 0, 0 )
WindowMaximized                 ( window, WindowMaximized, 0, 0, 0, 0, 0, 0 )
WindowMinimized                 ( window, WindowMinimized, 0, 0, 0, 0, 0, 0 )
WindowRedraw                    ( window, WindowRedraw, xWin, yWin, width, height, 0, 0 )
WindowResized                   ( window, WindowResized, xDisp, yDisp, width, height, 0, 0 )
WindowSelected                  ( window, WindowSelected, 0, 0, 0, 0, 0, 0 )
```

Miscellaneous Functions

```
Xgr                          ( )
XgrBorderNameToNumber        ( border$, @border )
XgrBorderNumberToName        ( border, @border$ )
XgrBorderNumberToWidth       ( border, @width )
XgrCreateFont                ( @font, fontName$, fontSize, fontWeight, fontItalic, fontAngle )
XgrCursorNameToNumber        ( cursorName$, @cursorNumber )
XgrCursorNumberToName        ( cursorNumber, @cursorName$ )
XgrDestroyFont               ( font )
XgrGetClipboard              ( clipboard, @clipType, @text$, @image[] )
XgrGetCursor                 ( @cursor )
XgrGetDisplaySize            ( display$, @width, @height, @borderWidth, @titleHeight )
XgrGetFontInfo               ( font, @fontName$, @fontSize, @fontWeight, @fontItalic, @fontAngle )
XgrGetFontMetrics            ( font, @maxCharWidth, @maxCharHeight, @ascent, @decent, @gap, @flags )
XgrGetFontNames              ( @count, @fontNames$[] )
XgrGetKeystateModify         ( state, @modify, @edit )
XgrGetTextImageSize          ( font, text$, @dx, @dy, @width, @height, @topGap, @flags )
XgrIconNameToNumber          ( iconName$, @iconNumber )
XgrIconNumberToName          ( iconNumber, @iconName$ )
XgrRegisterCursor            ( cursorName$, @cursor )
XgrRegisterIcon              ( iconName$, @icon )
XgrSetClipboard              ( clipboard, clipType, @text$, @image[] )
XgrSetCursor                 ( cursor, @oldCursor )
version$ = XgrVersion$       ( )
```

Xgr	**( )** **Xgr()** initializes *GraphicsDesigner*. Every program or library that calls a graphics function must call **Xgr()** before any others. Calling **Xgr()** more than once has no harmful effects.
XgrBorderNameToNumber	**(border$, @border)** **XgrBorderNameToNumber()** converts a border name into a border number appropriate for **XgrDrawBorder()** functions.
XgrBorderNumberToName	**(border, @border$)** **XgrBorderNumberToName()** converts a border number into a border name.
XgrBorderNumberToWidth	**(border, @width)** **XgrBorderNumberToWidth()** converts a border number into the width of the border in pixels.
XgrCreateFont	**(@font, fontName$, fontSize, fontWeight, fontItalic, fontAngle)** **XgrCreateFont()** creates a font of the specified **fontName$, fontSize, fontWeight, fontItalic, fontAngle,** and returns its **font** number. All other functions that specify a font pass **font** as an argument. **fontName$** is the typeface name, **fontSize** is ten times the point size, **fontWeight** is boldness from very thin to very heavy (0 to 1000), **fontItalic** is tilt from none to extreme (0 to 1000), and **fontAngle** is the angle the characters are rotated from horizontal in 1/10 degree units (1800 = 180 degrees = upside down). If **fontName$** cannot be found or the requested font cannot be created for any reason, **0** is returned in **font**. The default font is **0**, so the font number returned by **XgrCreateFont()** is always valid,

	though the characters may not look like those expected when the specified font is not found.
XgrCursorNameToNumber	**(cursorName$, @cursorNumber)** **XgrCursorNameToNumber()** returns the **cursorNumber** assigned to **cursorName$** by **XgrRegisterCursor()**.
XgrCursorNumberToName	**(cursorNumber, @cursorName$)** **XgrCursorNumberToName()** returns the **cursorName$** to which **XgrRegisterCursor()** assigned **cursorNumber**.
XgrDestroyFont	**(font)** **XgrDestroyFont()** destroys **font**. The same font number may subsequently be assigned to a new font, so programs should not use the font number of a destroyed font.
XgrGetClipboard	**(clipboard, @clipType, @text$, @image[])** **XgrGetClipboard()** returns the current contents of **clipboard** in **text$** and/or **image[]**, depending on the type of data in **clipboard**, which is returned in **clipType**. The interapplication clipboard, also called the system clipboard, is **clipboard=0**. **clipType = 0 = $$ClipboardTypeNone** **clipType = 1 = $$ClipboardTypeText** **clipType = 2 = $$ClipboardTypeImage**
XgrGetCursor	**(@cursor)** **XgrGetCursor()** returns the currently displayed **cursor**.
XgrGetDisplaySize	**(display$, @width, @height, @borderWidth, @titleHeight)** **XgrGetDisplaySize()** returns the **width,height** of **display$** in pixels, as well as the **borderWidth** and **titleHeight** of windows that have borders and title-bars. **display$ = ""** denotes the default display.
XgrGetFontInfo	**(font, @fontName$, @fontSize, @fontWeight, @fontItalic, @fontAngle)** **XgrGetFontInfo()** returns **fontName$, fontSize, fontWeight, fontItalic,** and **fontAngle** for **font**. See **XgrCreateFont()** for details.
XgrGetFontMetrics	**(font, @maxCharWidth, @maxCharHeight, @ascent, @decent, @gap, @flags)** **XgrGetFontMetrics()** returns the maximum character width in pixels, maximum character height in pixels, the ascent from the baseline of the tallest character, the decent from the baseline for the lowest character, the gap at the top that is normally interline spacing, but may contain active character pixels for unusual characters, including characters with accents and umlauts.

XgrGetFontNames	**(@fontName$[])** **XgrGetFontNames()** returns the names of all typefaces from which fonts can be created by **XgrCreateFont()**.
XgrGetKeystateModify	**(state, @modify, @edit)** **XgrGetKeystateModify()** *estimates* whether a **#KeyDown** message with the specified **state** argument would normally modify text in a common text grids like **XuiTextLine** and **XuiTextArea**.
XgrGetTextImageSize	**(font, text$, @dx, @dy, @width, @height, @gap, @flags)** **XgrGetTextImageSize()** computes the **(dx,dy)** change in drawpoint and the **(width,height)** of the smallest rectangle that contain the text image of **text$** when drawn with **font**. Since **font** includes a rotation attribute, the image of the text string may be tipped from the horizontal, shifting the drawpoint vertically as well as horizontally. This fact is reflected in **(dx,dy)**, but not **(width,height)** because the size of the smallest rectangle does not change as text strings are rotated.
XgrIconNameToNumber	**(iconName$, @iconNumber)** **XgrIconNameToNumber()** converts **iconName$** into the **iconNumber** originally assigned it by **XgrRegisterIcon()**. If **iconName$** was never registered, **iconNumber = 0**.
XgrIconNumberToName	**(iconNumber, @iconName$)** **XgrIconNumberToName()** converts **iconNumber** into the **iconName$** it was created for by **XgrRegisterIcon()**. If **iconNumber** has not been assigned to any icon by **XgrRegisterIcon(), iconName$ = ""**.
XgrRegisterCursor	**(cursorName$, @cursorNumber)** **XgrRegisterCursor()** assigns a unique **cursorNumber** for **cursorName$,** or returns its existing **cursorNumber** if **cursorName$** has already been registered.
XgrRegisterIcon	**(iconName$, @iconNumber)** **XgrRegisterIcon()** assigns a unique **iconNumber** for **iconName$,** or returns its existing **iconNumber** if **iconName$** has already been registered.
XgrSetClipboard	**(clipboard, clipType, @text$, @image[])** **XgrSetClipboard()** installs **text$** and/or **image[]** into **clipboard,** depending on the type of data specified by **clipType**. The system clipboard is **clipboard=0**. **clipType = 0 = $$ClipboardTypeNone** **clipType = 1 = $$ClipboardTypeText** **clipType = 2 = $$ClipboardTypeImage**

XgrSetCursor	**(cursor, @oldCursor)** **XgrSetCursor()** sets the current **cursor** and returns the **oldCursor**. The displayed cursor changes to **cursor**.
version$ = XgrVersion$	**( )** **XgrVersion$()** returns the current *GraphicsDesigner* **version$**.

Color Functions

```
XgrConvertColorToRGB          ( color, @red, @green, @blue )
XgrConvertRGBToColor          ( red, green, blue, @color )
XgrGetBackgroundColor         ( grid, @color )
XgrGetBackgroundRGB           ( grid, @red, @green, @blue )
XgrGetDefaultColors           (@back, @draw, @lo, @hi, @dull, @acc, @lowtext, @hightext )
XgrGetDrawingColor            ( grid, @color )
XgrGetDrawingRGB              ( grid, @red, @green, @blue )
XgrGetGridColors              ( grid, @back, @draw, @lo, @hi, @dull, @acc, @lowtext, @hightext )
XgrSetBackgroundColor         ( grid, color )
XgrSetBackgroundRGB           ( grid, red, green, blue )
XgrSetDefaultColors           ( back, draw, lo, hi, dull, acc, lowtext, hightext )
XgrSetDrawingColor            ( grid, color )
XgrSetDrawingRGB              ( grid, red, green, blue )
XgrSetGridColors              ( grid, back, draw, lo, hi, dull, acc, lowtext, hightext )
```

XgrConvertColorToRGB	**(color, @red, @green, @blue)** **XgrConvertColorToRGB()** converts **color** into 16-bit per color RGB format **(red,green,blue)** intensities.
XgrConvertRGBToColor	**(red, green, blue, @color)** **XgrConvertRGBToColor()** combines 16-bit per color RGB format **(red,green,blue)** intensities into a **color** suitable for passing to drawing functions.
XgrGetBackgroundColor	**(grid, @color)** **XgrGetBackgroundColor()** returns the current background **color** of grid as **(red,green,blue,colorNumber)**, each an 8-bit value.
XgrGetBackgroundRGB	**(grid, @red, @green, @blue)** **XgrGetBackgroundRGB()** returns the current background color of **grid** as 16-bit **(red, green, blue)** intensities.
XgrGetDefaultColors	**(@back, @draw, @low, @high, @dull, @acc, @lowtext, @hightext)** **XgrGetDefaultColors()** returns the colors assigned to grids when they are created by **XgrCreateGrid()**.

XgrGetDrawingColor	**(grid, @color)** **XgrGetDrawingColor()** returns the current drawing **color** of **grid** as 8-bit **(red,green,blue,colorNumber)**.
XgrGetDrawingRGB	**(grid, @red, @green, @blue)** **XgrGetDrawingRGB()** returns the current drawing color of **grid** as 16-bit **(red,green,blue)** intensities.
XgrGetGridColors	**(grid, @back, @draw, @lo, @hi, @dull, @acc, @lowtext, @hightext)** **XgrGetGridColors()** returns the current colors of grid in **(back,draw,lo,hi,dull,acc,lowtext,hightext)**, each a 32-bit value with 8-bits per **red,green,blue,colorNumber**.
XgrSetBackgroundColor	**(grid, color)** **XgrSetBackgroundColor()** sets the current background color of **grid** to 8-bit **(red,blue,green,colorNumber)** values in **color**.
XgrSetBackgroundRGB	**(grid, @red, @green, @blue)** **XgrSetBackgroundRGB()** sets the current background color of **grid** to 16-bit **(red,green,blue)** intensities.
XgrSetDefaultColors	**(back, draw, low, high, dull, acc, lowtext, hightext)** **XgrSetDefaultColors()** sets the colors assigned to grids when they are created by **XgrCreateGrid()**.
XgrSetDrawingColor	**(grid, color)** **XgrSetDrawingColor()** sets the current drawing **color** of **grid** to 8-bit **(red,green,blue,colorNumber)** values.
XgrSetDrawingRGB	**(grid, @red, @green, @blue)** **XgrSetDrawingRGB()** sets the current drawing color of **grid** to 16-bit **(red,green,blue)** intensities.
XgrSetGridColors	**(grid, back, draw, lo, hi, dull, acc, lowtext, hightext)** **XgrSetGridColors()** sets the current colors of grid to **(back,draw,lo,hi,dull,acc,lowtext,hightext)**, each an 8-bit per **(red,green,blue,colorNumber)** value.

Window Functions

```
XgrClearWindow                ( window, color )
XgrClearWindowAndImages       ( window, color )
XgrCreateWindow               (@window, winType, @xDisp, @yDisp, @width, @height, winFunc, display$ )
XgrDestroyWindow              ( window )
XgrDisplayWindow              ( window )
XgrGetWindowFunction          ( window, @func )
XgrGetWindowIcon              ( window, @icon )
XgrGetWindowPositionAndSize   ( window, @xDisp, @yDisp, @width, @height )
XgrGetWindowState             ( window, @state )
XgrGetWindowTitle             ( window, @title$ )
XgrHideWindow                 ( window )
XgrMaximizeWindow             ( window )
XgrMinimizeWindow             ( window )
XgrRestoreWindow              ( window )
XgrSetWindowFunction          ( window, func )
XgrSetWindowIcon              ( window, icon )
XgrSetWindowPositionAndSize   ( window, xDisp, yDisp, width, height )
XgrSetWindowState             ( window, state )
XgrSetWindowTitle             ( window, title$ )
XgrShowWindow                 ( window )
```

XgrClearWindow	**(window, color)** **XgrClearWindow()** clears **window** to **color**. If **(color=-1)**, the window is cleared to the same color as last time, or to black if it has not been cleared previously. **XgrClearWindow()** does not clear image grids associated with the window.
XgrClearWindowAndImages	**(window, color)** **XgrClearWindowAndImages()** clears **window** to **color**. If **(color=-1)**, the window is cleared to the same color as last time, or to black if it has not been cleared previously. All image grids associated with a grid in the window are cleared to their current background color.

XgrCreateWindow `( @window, @winType, xDisp, yDisp, width, height, winFunc, display$ )`

XgrCreateWindow() creates and displays a new **winType** graphics window, and returns its window number in **window,** which will be no larger then the number of windows currently defined in the system. **winType** contains window type bits in the upper 16-bits and parent window in the low 16-bits. The low 16-bits are usually zero because most windows do not have a parent window. One exception is the pulldown list windows displayed by menu bars.

The location of the graphics window on the display can be requested by setting **(xDisp,yDisp)** to coordinates within the display boundaries. **-1** for **xDisp** or **yDisp** specifies no preference for that axis. **(xDisp,yDisp)** values determines window placement to the extent supported by the window system.

If available from the window system, **(xDisp,yDisp)** are returned with the display coordinates of the upper left corner of the ***drawable*** area of the window - within the resize frame and title-bar.

(width,height) request the drawable size of the window in pixels. **(width,height)** determines the size of windows to the extent supported by the window system. Requests that extend a window beyond the limits of the display are generally not honored. If available from the window system, **(width,height)** are returned with the size of the drawable area of the window.

In systems where windows can be directed to more than one display, **display$** is the name of the target display screen, like **"max:0.0"**.

If the window system represents displays with numbers, **display$** is a string form of the number, as in **"32"** or **"0x4F7D192C"**.

disp$ = "" denotes the default display.

```
*** window type *** - *** characteristics ***

$$WindowTopMost     - stays above other windows
$$WindowNoSelect    - window is not selected by mouse button events
$$WindowNoFrame     - window has no resize frame
$$WindowResizeFrame - window has a resize frame
$$WindowTitleBar    - window has a title bar to display a window name
$$WindowSystemMenu  - window has a system menu button
$$WindowMinimizeBox - window has a minimize button
$$WindowMaximizeBox - window has a maximize button
```

XgrDestroyWindow `( window )`

XgrDestroyWindow() destroys **window** after destroying its grids. **window** is then removed from the display and a **WindowDestroyed** message is added to the message queue. If **window** has child windows, they and their grids are destroyed and additional WindowDestroyed message is added to the message queue for each child window. All information about the window and its grids is permanently lost.

XgrDisplayWindow `( window )`

	XgrDisplayWindow() displays **window** if it hidden or inconified, then selects **window** if it is not already selected. Displaying **window** with **XgrDisplayWindow()** is the same as selecting **window** with **XgrSetSelectedWindow()**. **window** is selected and displayed above all others. The window argument of subsequent keyboard messages will be **window**.
XgrGetWindowFunction	**(window, @func)** **XgrGetWindowFunction()** returns in **func** the address of the window function assigned to **window**. **func** is the address function called by **XgrProcessMessages()** when it processes messages for **window** or a grid in **window**.
XgrGetWindowIcon	**(window, @icon)** **XgrGetWindowIcon()** returns the image of the **icon** displayed for **window** when it is minimized.
XgrGetWindowPositionAndSize	**(window, @xDisp, @yDisp, @width, @height)** **XgrGetWindowPositionAndSize()** returns the display coordinates of **window** on the display in **(xDisp,yDisp)**, and its **(width,height)** in pixels.
XgrGetWindowState	**(window, @state)** **XgrGetWindowState()** returns the current display **state** of **window**. **state=0,1,2,3** for hidden, displayed, minimized, maximized.
XgrGetWindowTitle	**(window, @title$)** **XgrGetWindowTitle()** returns the **title$** of **window**. **title$** is displayed on the title-bar of windows with title-bars.
XgrHideWindow	**(window)** **XgrHideWindow()** makes **window** invisible until it is made visible again by **XgrDisplayWindow()**.

XgrMaximizeWindow	**(window)** **XgrMaximizeWindow()** displays **window** as the largest size it can, up to the size of the display screen.
XgrMinimizeWindow	**(window)** **XgrMinimizeWindow()** displays **window** as a small icon in the lower part of the display. The icon can be double clicked to restore the window to its previously displayed state.
XgrRestoreWindow	**(window)** **XgrRestoreWindow()** displays window in its most recently displayed state (normal size, minimized, maximized).
XgrSetWindowFunction	**(window, func)** **XgrSetWindowFunction()** sets the address of the window function of **window** to **func**. **func** is the address called by **XgrProcessMessages()** when it processes a message for **window** or a grid in **window**.
XgrSetWindowIcon	**(window, @icon)** **XgrSetWindowIcon()** sets the image of the **icon** displayed for **window** when it is minimized.
XgrSetWindowPositionAndSize	**(window, xDisp, yDisp, width, height)** **XgrSetWindowPositionAndSize()** sets the position of **window** on the display to **(xDisp,yDisp),** and resizes it to **(width,height)** pixels if necessary.
XgrSetWindowState	**(window, state)** **XgrSetWindowState()** sets the current display state of **window** to **state**. **state=0,1,2,3** for hidden, displayed, minimized, maximized.
XgrSetWindowTitle	**(window, title$)** **XgrSetWindowTitle()** sets the **title$** of **window**. **title$** is displayed on the title-bar of windows with title-bars.
XgrShowWindow	**(window)** **XgrShowWindow()** displays **window** without selecting it. If **window** cannot be displayed without selecting, it is displayed

Grid Functions

```
XgrClearGrid                    ( grid, color )
XgrConvertDisplayToGrid         ( grid, xDisp, yDisp, @xGrid, @yGrid )
XgrConvertDisplayToLocal        ( grid, xDisp, yDisp, @x, @y )
XgrConvertDisplayToScaled       ( grid, xDisp, yDisp, @x#, @y# )
XgrConvertDisplayToWindow       ( grid, xDisp, yDisp, @xWin, @yWin )
XgrConvertGridToDisplay         ( grid, xGrid, yGrid, @xDisp, @yDisp )
XgrConvertGridToLocal           ( grid, xGrid, yGrid, @x, @y )
XgrConvertGridToScaled          ( grid, xGrid, yGrid, @x#, @y# )
XgrConvertGridToWindow          ( grid, xGrid, yGrid, @xWin, @yWin )
XgrConvertLocalToDisplay        ( grid, x, y, @xDisp, @yDisp )
XgrConvertLocalToGrid           ( grid, x, y, @xGrid, @yGrid )
XgrConvertLocalToScaled         ( grid, x, y, @x#, @y# )
XgrConvertLocalToWindow         ( grid, x, y, @xWin, @yWin )
XgrConvertScaledToDisplay       ( grid, x#, y#, @xDisp, @yDisp )
XgrConvertScaledToGrid          ( grid, x#, y#, @xGrid, @yGrid )
XgrConvertScaledToLocal         ( grid, x#, y#, @x, @y )
XgrConvertScaledToWindow        ( grid, x#, y#, @xWin, @yWin )
XgrConvertWindowToDisplay       ( grid, xWin, yWin, @xDisp, @yDisp )
XgrConvertWindowToGrid          ( grid, xWin, yWin, @xGrid, @yGrid )
XgrConvertWindowToLocal         ( grid, xWin, yWin, @x, @y )
XgrConvertWindowToScaled        ( grid, xWin, yWin, @x#, @y# )
XgrCreateGrid                   (@grid, gridType, xWin, yWin, width, height, window, parent, func )
XgrDestroyGrid                  ( grid )
XgrGetGridBorder                ( grid, @border, @borderUp, @borderDown, @borderFlags )
XgrGetGridBorderOffset          ( grid, @left, @top, @right, @bottom )
XgrGetGridBoxGrid               ( grid, @x1Grid, @y1Grid, @x2Grid, @y2Grid )
XgrGetGridBoxLocal              ( grid, @x1, @y1, @x2, @y2 )
XgrGetGridBoxScaled             ( grid, @x1#, @y1#, @x2#, @y2# )
XgrGetGridBoxWindow             ( grid, @x1Win, @y1Win, @x2Win, @y2Win )
XgrGetGridBuffer                ( grid, @bufferGrid )
XgrGetGridCharacterMapArray     ( grid, @map[] )
XgrGetGridCoords                ( grid, @xParent, @yParent, @x1Grid, @y1Grid, @x2Grid, @y2Grid )
XgrGetGridDrawingMode           ( grid, @drawingMode, @lineStyle, @lineWidth )
XgrGetGridFont                  ( grid, @font )
XgrGetGridFunction              ( grid, @func )
XgrGetGridParent                ( grid, @parent )
XgrGetGridPositionAndSize       ( grid, @xWin, @yWin, @width, @height )
XgrGetGridState                 ( grid, @state )
XgrGetGridType                  ( grid, @gridType )
XgrGetGridWindow                ( grid, @window )
XgrGridTypeNameToNumber         ( gridType$, @gridType )
XgrGridTypeNumberToName         ( gridType, @gridType$ )
XgrRegisterGridType             ( gridType$, @gridType )
XgrSetGridBorder                ( grid, border, borderUp, borderDown, borderFlags )
XgrSetGridBorderOffset          ( grid, left, top, right, bottom )
XgrSetGridBoxGrid               ( grid, x1Grid, y1Grid, x2Grid, y2Grid )
XgrSetGridBoxScaled             ( grid, x1#, y1#, x2#, y2# )
XgrSetGridBuffer                ( grid, bufferGrid )
XgrSetGridCharacterMapArray     ( grid, @map[] )
XgrSetGridDrawingMode           ( grid, drawingMode, lineStyle, lineWidth )
XgrSetGridFont                  ( grid, font )
XgrSetGridFunction              ( grid, funcAddr )
XgrSetGridPositionAndSize       ( grid, xWin, yWin, width, height )
XgrSetGridState                 ( grid, state )
XgrSetGridTimer                 ( grid, msTimeInterval )
XgrSetGridType                  ( grid, gridType )
```

XgrClearGrid	**(grid, color)** **XgrClearGrid()** clears **grid** to **color,** or to the current background color if **(color=-1)**. If **grid** is an image grid, or an image grid is attached to **grid,** it is cleared.
XgrConvertDisplayToGrid	**(grid, xDisp, yDisp, @xGrid, @yGrid)** **XgrConvertDisplayToGrid()** converts display coordinates **(xDisp,yDisp)** to grid coordinates **(xGrid,yGrid)** for grid.
XgrConvertDisplayToLocal	**(grid, xDisp, yDisp, @x, @y)** **XgrConvertDisplayToGrid()** converts display coordinates **(xDisp,yDisp)** to local coordinates **(x,y)** for grid.
XgrConvertDisplayToScaled	**(grid, xDisp, yDisp, @x#, @y#)** **XgrConvertDisplayToScaled()** converts display coordinates **(xDisp,yDisp)** to scaled coordinates **(x#,y#)** for **grid**.
XgrConvertDisplayToWindow	**(grid, xDisp, yDisp, @xWin, @yWin)** **XgrConvertDisplayToWindow()** converts display coordinates **(xDisp,yDisp)** to window coordinates **(xWin,yWin)** for the window that contains **grid**.
XgrConvertGridToDisplay	**(grid, xGrid, yGrid, @xDisp, @yDisp)** **XgrConvertGridToDisplay()** converts grid coordinates **(xGrid,yGrid)** in **grid** to display coordinates **(xDisp,yDisp)**.
XgrConvertGridToLocal	**(grid, xGrid, yGrid, @x, @y)** **XgrConvertGridToLocal()** converts grid coordinates **(xGrid,yGrid)** in **grid** to **(x,y)** local coordinates.
XgrConvertGridToScaled	**(grid, xGrid, yGrid, @x#, @y#)** **XgrConvertGridToScaled()** converts grid coordinates **(xGrid,yGrid)** to scaled coordinates **(x#,y#)** for **grid**.
XgrConvertGridToWindow	**(grid, xGrid, yGrid, @xWin, @yWin)** **XgrConvertGridToWindow()** converts grid coordinates **(xGrid,yGrid)** in **grid** to **(xWin,yWin)** window coordinates.

XgrConvertLocalToDisplay	**(grid, x, y, @xDisp, @yDisp)** **XgrConvertGridToDisplay()** converts grid coordinates **(x,y)** in **grid** to display coordinates **(xDisp,yDisp)**.
XgrConvertLocalToGrid	**(grid, x, y, @xGrid, @yGrid)** **XgrConvertGridToLocal()** converts grid coordinates **(x,y)** in **grid** to **(xGrid,yGrid)** local coordinates.
XgrConvertLocalToScaled	**(grid, x, y, @x#, @y#)** **XgrConvertGridToScaled()** converts grid coordinates **(x,y)** to scaled coordinates **(x#,y#)** for **grid**.
XgrConvertLocalToWindow	**(grid, x, y, @xWin, @yWin)** **XgrConvertGridToWindow()** converts grid coordinates **(xGrid,yGrid)** in **grid** to **(xWin,yWin)** window coordinates.
XgrConvertScaledToDisplay	**(grid, x#, y#, @xDisp, @yDisp)** **XgrConvertScaledToDisplay()** converts scaled coordinates **(x#,y#)** in **grid** to display coordinates **(xDisp,yDisp)**.
XgrConvertScaledToGrid	**(grid, x#, y#, @xGrid, @yGrid)** **XgrConvertScaledToGrid()** converts scaled coordinates **(x#,y#)** into grid coordinates **(xGrid,yGrid)** for **grid**.
XgrConvertScaledToLocal	**(grid, x#, y#, @x, @y)** **XgrConvertScaledToLocal()** converts scaled coordinates **(x#,y#)** into local coordinates **(x,y)** for **grid**.
XgrConvertScaledToWindow	**(grid, x#, y#, @xWin, @yWin)** **XgrConvertScaledToWindow()** converts scaled coordinates **(x#,y#)** in **grid** to window coordinates **(xWin,yWin)**.

XgrConvertWindowToDisplay	**(grid, xWin, yWin, @xDisp, @yDisp)** **XgrConvertWindowToDisplay()** converts window coordinates **(xWin,yWin)** to display coordinates **(xDisp,yDisp)** for the window that contains **grid**.
XgrConvertWindowToGrid	**(grid, xWin, yWin, @xGrid, @yGrid)** **XgrConvertWindowToGrid()** converts window coordinates **(xWin,yWin)** to **(xGrid,yGrid)** grid coordinates for **grid**.
XgrConvertWindowToLocal	**(grid, xWin, yWin, @x, @y)** **XgrConvertWindowToGrid()** converts window coordinates **(xWin,yWin)** to **(x,y)** local coordinates for **grid**.
XgrConvertWindowToScaled	**(grid, xWin, yWin, @x#, @y#)** **XgrConvertWindowToScaled()** converts window coordinates **(xWin,yWin)** into **(x#,y#)** scaled coordinates for **grid**.
XgrCreateGrid	**(@grid, gridType, x, y, width, height, window, parent, func)** **XgrCreateGrid()** creates a new grid in **window,** returns its **grid** number, initializes its settings to default values, then initializes its **(gridType,parent,func)** settings. The upper-left corner of **grid** is set to: **(x,y) Local Coordinates of parent grid/window (locates grid-box)** **(0,0) Local Coordinates** **(0,0) Grid Coorinates** **(0#,0#) Scaled Coordinates** while the lower-right corner of grid is set to: **(width-1, height-1) Local Coordinates** **(width-1, height-1) Grid Coordinates** **(1#, 1#) Scaled Coordinates**
XgrDestroyGrid	**(grid)** **XgrDestroyGrid()** destroys **grid,** and make the **grid** number available. If **grid** is **bufferGrid** in other grids, **bufferGrid** is set to **0** in those grids. If **grid** is an image grid type, its memory image is freed.
XgrGetGridBorder	**(grid, @border, @borderUp, @borderDown, @borderFlags)** **XgrGetGridBorder()** returns the current grid **border** attribute that specifies the border style in the **XgrDrawBorder()** functions, plus additional border values **borderUp**, **borderDown**, **borderFlags**.
XgrGetGridBorderOffset	**(grid, @left, @top, @right, @bottom)** **XgrGetGridBorderOffset()** returns the current border offset from the **left**, **top**, **right**, and **bottom** of grid that controls where the border is drawn by the **XgrDrawBorder()** functions.
XgrGetGridBoxDisplay	**(grid, @x1Disp, @y1Disp, @x2Disp, @y2Disp)**

	XgrGetGridBox() returns the display coordinates of the upper-left and lower-right corners of the grid-box for **grid** in **(x1Disp,y1Disp:x2Disp,y2Disp)**.
XgrGetGridBoxGrid	**(grid, @x1Grid, @y1Grid, @x2Grid, @y2Grid)** **XgrGetGridBoxGrid()** returns the grid coordinates of the upper-left and lower-right corners of the grid-box for **grid** in **(x1Grid,y1Grid:x2Grid,y2Grid)**.
XgrGetGridBoxLocal	**(grid, @x1, @y1, @x2, @y2)** **XgrGetGridBoxLocal()** returns local coordinates of the upper-left and lower-right corners of the grid-box for **grid** in **(x1,y1:x2,y2)**.
XgrGetGridBoxScaled	**(grid, x1#, y1#, x2#, y2#)** **XgrGetGridBoxScaled()** returns the scaled coordinates of the upper-left and lower-right corners of the grid-box for **grid** in **(x1#,y1#:x2#,y2#)**.
XgrGetGridBoxWindow	**(grid, @x1Win, @y1Win, @x2Win, @y2Win)** **XgrGetGridBoxWindow()** returns the window coordinates of the upper-left and lower-right corners of the grid-box for **grid** in **(x1Win,y1Win:x2Win,y2Win)**.
XgrGetGridBuffer	**(grid, @bufferGrid)** **XgrGetGridBuffer()** returns in **bufferGrid** the image grid that is currently performing automatic buffering for **grid**. **bufferGrid=0** if automatic buffering is not being performed.
XgrGetGridCharacterMapArray()	**(grid, @map[])** **XgrGetGridCharacterMapArray()** returns a copy of the character map array for the specified **grid**.
XgrGetGridCoords	**(grid, @x, @y, @x1, @y1, @x2, @y2)** **XgrGetGridCoords()** returns parents local coordinates of the upper-left corner of **grid** in **x,y**, and the local coordinates of the **(upper-left:lower-right)** corners of **grid** in **(x1,y1:x2,y2)**.
XgrGetGridDrawingMode	**(grid, @drawingMode, @lineStyle, @lineWidth)** **XgrGetDrawingMode()** returns the current **drawingMode** for **grid**, including **(lineStyle,lineWidth)**. **0** in **drawingMode** means **SET** drawing mode, where pixels are drawn with the specified color. **1** means **XOR** drawing mode, where pixels are drawn with the color generated by a bitwise **XOR** of the current pixel color and the specified color. **lineWidth, lineStyle** may not be implemented.
XgrGetGridFont	**(grid, @font)**

	XgrGetGridFont() returns the **font** currently assigned to **grid**. **font** determines the typeface, character size, and drawing angle for text drawn in **grid**.
XgrGetGridFunction	**(grid, @func)** **XgrGetGridFunction()** returns the **func** address of the grid function currently assigned to **grid**. This is the function called by **XgrProcessMessages()** when it processes grid messages for **grid**.
XgrGetGridParent	**(grid, @parent)** **XgrGetGridParent()** returns the **parent** currently assigned to **grid**. Grids with **parent=0** have no parent. Parentless grids are the only grids whose grid functions are called and sent **Redraw** messages by **XgrRedrawWindow()**.
XgrGetGridPositionAndSize	**(grid, @x, @y, @width, @height)** **XgrGetGridPositionAndSize()** returns the local coordinates in its parent of the upper-left corner of the grid-box of **grid** in **(x,y)**, and the width and height of the grid-box in **(width,height)**.
XgrGetGridState	**(grid, @state)** **XgrGetGridState()** returns the **state** currently assigned to **grid**. **state=0** disables **grid**. *GraphicsDesigner* does not consider disabled grids in its search to find the grid to send mouse messages to, which effectively makes disabled grids invisible or non-existant to the mouse.
XgrGetGridType	**(grid, @gridType)** **XgrGetGridType()** returns the **gridType** of **grid**.
XgrGetGridWindow	**(grid, @window)** **XgrGetGridWindow()** returns the **window** that contains **grid**.

XgrGridTypeNameToNumber	**(gridType$, @gridType)** **XgrGridTypeNameToNumber()** returns the **gridType** number previously assigned to **gridType$**. If **gridType$** was never registered, **-1** is returned in **gridType**.
XgrGridTypeNumberToName	**(gridType, @gridType$)** **XgrGridTypeNumberToName()** converts **gridType** into the corresponding **gridType$**. If **gridType** has not been assigned, an empty string is returned in **gridType$**. **gridType=0** is "Coordinate" and **gridType=1** is "Image".
XgrRegisterGridType	**(gridType$, @gridType)** **XgrRegisterGridType()** registers **gridType$**, assigns it a grid type number, and returns it in **gridType**. If **gridType$** already exists, **XgrRegisterGridType()** returns the already established grid type number.
XgrSetGridBorder	**(grid, border, borderUp, borderDown, borderFlags)** **XgrSetGridBorder()** sets the current grid **border** attribute that specifies the border style in the **XgrDrawBorder()** functions, plus additional border values **borderUp**, **borderDown**, **borderFlags**.
XgrSetGridBorderOffset	**(grid, left, top, right, bottom)** **XgrSetGridBorderOffset()** sets the current border offset from the **left**, **top**, **right**, and **bottom** of grid. These values determine how far the grid border is drawn from the edge of the grid by the **XgrDrawBorder()** functions.
XgrSetGridBoxGrid	**(grid, x1Grid, y1Grid, x2Grid, y2Grid)** **XgrSetGridBox()** sets the **(upper-left:lower-right)** of the grid-box to grid coordinates **(x1Grid,y1Grid:x2Grid,y2Grid)**. This does not change the position or size of the grid, and values of **x2Grid,y2Grid** that are inconsistent with grid width and height are adjusted to make them so.
XgrSetGridBoxScaled	**(grid, x1#, y1#, x2#, y2#)** **XgrSetGridBoxScaled()** sets the **(upper-left:lower-right)** corners of the grid-box to scaled coordinates **(x1#,y1#:x2#,y2#)**. This does not change the position or size of the grid.
XgrSetGridBuffer	**(grid, bufferGrid)** **XgrSetGridBuffer()** sets **bufferGrid** as the image grid to perform automatic buffering for **grid**. The grid type of **bufferGrid** must be image grid, which is **1**. **bufferGrid = 0** means no buffer.
XgrSetGridCharacterMapArray()	**(grid, @map[])**

	XgrSetGridCharacterMapArray() sets the character map array for the specified **grid**.
XgrSetGridDrawingMode	**(grid, drawingMode, lineWidth, lineStyle)** **XgrSetGridDrawingMode()** sets **drawMode** for **grid**. **0** in **drawingMode** means **SET** drawing mode, where pixels are drawn with the specified color. **1** means **XOR** drawing mode, where pixels are drawn with the color generated by a bitwise **XOR** of the current pixel color and the specified color. **lineWidth,lineStyle** may not be implemented.
XgrSetGridFont	**(grid, font)** **XgrSetGridFont()** sets the **font** that controls the typeface, point size, and drawing angle of text drawn in **grid**.
XgrSetGridFunction	**(grid, func)** **XgrSetGridFunction()** sets the **func** address of the grid function assigned to **grid**. **XgrProcessMessages()** calls this function when it processes a message for **grid**.
XgrSetGridPositionAndSize	**(grid, x, y, width, height)** **XgrSetGridPositionAndSize()** positions and resizes **grid**. The upper-left corner of its grid-box is set to parent local coordinates **(x,y)**, and its **(x2Grid,y2Grid)** grid coordinates are adjusted if the size is changed.
XgrSetGridState	**(grid, state)** **XgrSetGridState()** sets the disable/enable state of **grid** to **state**. **state=0** disables grids. *GraphicsDesigner* does not consider disabled grids in its search to find the grid to send mouse messages to, which effectively makes disabled grids invisible or non-existant to the mouse.
XgrSetGridTimer	**(grid, interval)** **XgrSetGridTimer()** sets a millisecond timer for **grid** to **interval**. When the timer counts down to zero it stops and a **Timer** message for **grid** is added to the message queue.
XgrSetGridType	**(grid, gridType)** **XgrSetGridType()** sets the **gridType** of **grid**. This has no effect on the functionality of *GraphicsDesigner*, except that **gridType=0** and **gridType=1** grids do not receive event messages.

Drawing Functions

```
XgrDrawArc                    ( grid, color, r, startAngle#, endAngle# )
XgrDrawArcGrid                ( grid, color, r, startAngle#, endAngle# )
XgrDrawArcScaled              ( grid, color, r#, startAngle#, endAngle# )
XgrDrawBorder                 ( grid, border, back, low, high, x1, y1, x2, y2 )
XgrDrawBorderGrid             ( grid, border, back, low, high, x1Grid, y1Grid, x2Grid, y2Grid )
XgrDrawBorderScaled           ( grid, border, back, low, high, x1#, y1#, x2#, y2# )
XgrDrawBox                    ( grid, color, x1, y1, x2, y2 )
XgrDrawBoxGrid                ( grid, color, x1Grid, y1Grid, x2Grid, y2Grid )
XgrDrawBoxScaled              ( grid, color, x1#, y1#, x2#, y2# )
XgrDrawCircle                 ( grid, color, r )
XgrDrawCircleGrid             ( grid, color, r )
XgrDrawCircleScaled           ( grid, color, r# )
XgrDrawGridBorder             ( grid, border )
XgrDrawIcon                   ( grid, icon, x, y )
XgrDrawIconGrid               ( grid, icon, xGrid, yGrid )
XgrDrawIconScaled             ( grid, icon, x#, y# )
XgrDrawLine                   ( grid, color, x1, y1, x2, y2 )
XgrDrawLineGrid               ( grid, color, x1Grid, y1Grid, x2Grid, y2Grid )
XgrDrawLineScaled             ( grid, color, x1#, y1#, x2#, y2# )
XgrDrawLineTo                 ( grid, color, x, y )
XgrDrawLineToGrid             ( grid, color, xGrid, yGrid )
XgrDrawLineToScaled           ( grid, color, x#, y# )
XgrDrawLineToDelta            ( grid, color, dx, dy )
XgrDrawLineToDeltaGrid        ( grid, color, dxGrid, dyGrid )
XgrDrawLineToDeltaScaled      ( grid, color, dx#, dy# )
XgrDrawLines                  ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesGrid              ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesScaled            ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesTo                ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesToGrid            ( grid, color, first, count, ANY @lines[] )
XgrDrawLinesToScaled          ( grid, color, first, count, ANY @lines[] )
XgrDrawPoint                  ( grid, color, x, y )
XgrDrawPointGrid              ( grid, color, xGrid, yGrid )
XgrDrawPointScaled            ( grid, color, x#, y# )
XgrDrawPoints                 ( grid, color, first, count, ANY @points[] )
XgrDrawPointsGrid             ( grid, color, first, count, ANY @points[] )
XgrDrawPointsScaled           ( grid, color, first, count, ANY @points[] )
XgrDrawText                   ( grid, color, text$ )
XgrDrawTextGrid               ( grid, color, text$ )
XgrDrawTextScaled             ( grid, color, text$ )
XgrDrawTextFill               ( grid, color, text$ )
XgrDrawTextFillGrid           ( grid, color, text$ )
XgrDrawTextFillScaled         ( grid, color, text$ )
XgrFillBox                    ( grid, color, x1, y1, x2, y2 )
XgrFillBoxGrid                ( grid, color, x1Grid, y1Grid, x2Grid, y2Grid )
XgrFillBoxScaled              ( grid, color, x1#, y1#, x2#, y2# )
XgrGetDrawpoint               ( grid, @x, @y )
XgrGetDrawpointGrid           ( grid, @xGrid, @yGrid )
XgrGetDrawpointScaled         ( grid, @x#, @y# )
XgrGrabPoint                  ( grid, x, y, @red, @green, @blue, @color )
XgrGrabPointGrid              ( grid, xGrid, yGrid, @red, @green, @blue, @color )
XgrGrabPointScaled            ( grid, x#, y#, @red, @green, @blue, @color )
XgrMoveDelta                  ( grid, dx, dy )
XgrMoveDeltaGrid              ( grid, dxGrid, dyGrid )
XgrMoveDeltaScaled            ( grid, dx#, dy# )
XgrMoveTo                     ( grid, x, y )
XgrMoveToGrid                 ( grid, xGrid, yGrid )
XgrMoveToScaled               ( grid, x#, y# )
XgrSetDrawpoint               ( grid, x, y )
XgrSetDrawpointGrid           ( grid, xGrid, yGrid )
XgrSetDrawpointScaled         ( grid, x#, y# )
```

XgrDrawArc **XgrDrawArcGrid** **XgrDrawArcScaled**	**(grid, color, r, startAngle#, endAngle#)** **(grid, color, rGrid, startAngle#, endAngle#)** **(grid, color, r#, startAngle#, endAngle#)** These functions draw an arc, its center of curvature at the appropriate drawpoint, which need not be within the grid. The arc itself is **r** pixels, **rGrid** pixels or **r#** scaled units from the center of curvature. Drawing begins at **startAngle#** and ends at **endAngle#,** both of which are expressed in radians. Angles increase counterclockwise, and a circle is **$$TWOPI** radians. Angles are folded into the range **0** to **$$TWOPI** before drawing. **color = -1** means draw in the current drawing color. No drawpoint is changed. No color attribute is changed.
XgrDrawBorder **XgrDrawBorderGrid** **XgrDrawBorderScaled**	**(grid, border, back, low, high, x1, y1, x2, y2)** **(grid, border, back, low, high, x1Grid, y1Grid, x2Grid, y2Grid)** **(grid, border, back, low, high, x1#, y1#, x2#, y2#)** These functions draw a border at the specified coordinates. **border, back, low, high = -1** means draw with current value. No drawpoint is changed. No border or color attribute is changed.
XgrDrawBox **XgrDrawBoxGrid** **XgrDrawBoxScaled**	**(grid, color, x1, y1, x2, y2)** **(grid, color, x1Grid, y1Grid, x2Grid, y2Grid)** **(grid, color, x1#, y1#, x2#, y2#)** These functions draw a rectangle at the specified coordinates. **color = -1** means draw in the current drawing color. No drawpoint is changed. No color attribute is changed.
XgrDrawCircle **XgrDrawCircleGrid** **XgrDrawCircleScaled**	**(grid, color, r)** **(grid, color, rGrid)** **(grid, color, r#)** These functions draw a circle centered at the appropriate drawpoint. **color = -1** means draw in the current drawing color. No drawpoint is changed. No color attribute is changed.
XgrDrawGridBorder	**(grid, border)** **XgrDrawGridBorder()** draws the specified border at the current border offsets. **border = -1** means draw with the current border attribute. No drawpoint or border attribute is changed.
XgrDrawIcon	**(grid, icon, x, y)**

XgrDrawIconGrid **XgrDrawIconScaled**	**(grid, icon, xGrid, yGrid)** **(grid, icon, x#, y#)** These functions draw **icon** at the specified coordinates. **color = -1** means draw in the current drawing color. No drawpoint is changed. No color attribute is changed.

XgrDrawLine **XgrDrawLineGrid** **XgrDrawLineScaled**	**(grid, color, x1, y1, x2, y2)** **(grid, color, x1Grid, y1Grid, x2Grid, y2Grid)** **(grid, color, x1#, y1#, x2#, y2#)** These functions draw a line between the two specified points. Then the appropriate drawpoint is set to the second point. **color = -1** means draw in the current drawing color. No color attribute is changed.
XgrDrawLineTo **XgrDrawLineToGrid** **XgrDrawLineToScaled**	**(grid, color, x, y)** **(grid, color, xGrid, yGrid)** **(grid, color, x#, y#)** These functions draw a line from the appropriate drawpoint to the specified coordinates. Then the appropriate drawpoint is set to the specified coordinates. **color = -1** means draw in the current drawing color. No color attribute is changed.
XgrDrawLineToDelta **XgrDrawLineToDeltaGrid** **XgrDrawLineToDeltaScaled**	**(grid, color, dx, dy)** **(grid, color, dxGrid, dyGrid)** **(grid, color, dx#, dy#)** These functions draw a line from the appropriate drawpoint to an endpoint offset from the drawpoint by the specified values. Then the appropriate drawpoint is set to the endpoint. **color = -1** means draw in the current drawing color. No color attribute is changed.

XgrDrawLines **XgrDrawLinesGrid** **XgrDrawLinesScaled**	**(grid, color, first, count, @lines[])** **(grid, color, first, count, @linesGrid[])** **(grid, color, first, count, @lines#[])** These functions draw a series of independent line segments. Then the appropriate drawpoint is set to the endpoint of the last drawn line. Each line occupies four array elements. The first four array elements are the start and end point of line #0. Since the array contains the start point and end point of every line, every line is independent, and not necessarily connected to any previous or subsequent line. **first** is the first line to draw, starting at line #0. **count** is the number of lines to draw. **count = 0** means draw lines until the array is exhausted. **color = -1** means draw in the current drawing color. No color attribute is changed.
XgrDrawLinesTo **XgrDrawLinesToGrid** **XgrDrawLinesToScaled**	**(grid, color, first, count, @points[])** **(grid, color, first, count, @pointsGrid[])** **(grid, color, first, count, @points#)** These functions draw a series of connected lines between points. Then the appropriate drawpoint is set to the end point of the last drawn line. Each point occupies two array elements. The first two array elements are the start point of the first line. A move to the **first** point is performed, then **count** lines are drawn to successive points, or until the array of points is exhausted, whichever comes first. Since each point in the array is the end point of one line and the start point of the next, these functions draw a series of connected lines. **first** is the start point of the first line to draw, starting at point #0. **count** is the number of lines to draw. **count = 0** means draw lines until the array is exhausted. **color = -1** means draw in the current drawing color. No color attribute is changed.

XgrDrawPoint **XgrDrawPointGrid** **XgrDrawPointScaled**	**(grid, color, x, y)** **(grid, color, xGrid, yGrid)** **(grid, color, x#, y#)** These functions draw a point at the specified coordinates. Then the appropriate drawpoint is set to the specified coordinates. **color = -1** means draw in the current drawing color. No color attribute is changed.
XgrDrawPoints **XgrDrawPointsGrid** **XgrDrawPointsScaled**	**(grid, color, first, count, @points[])** **(grid, color, first, count, @pointsGrid[])** **(grid, color, first, count, @points#[])** These functions draw a series of points. Then the appropriate drawpoint is set to the last point drawn. Each point occupies two array elements. The first two array elements are point #0. **first** is the first point drawn. Successive points are drawn until the array is exhausted or **count** points have been drawn. **count = 0** means draw points until the array is exhausted. **color = -1** means draw in the current drawing color. No color attribute is changed.
XgrDrawText **XgrDrawTextGrid** **XgrDrawTextScaled**	**(grid, color, text$)** **(grid, color, text$)** **(grid, color, text$)** These functions draw text at the appropriate drawpoint without changing the pixels around the characters. Then the appropriate drawpoint is left after the last character. **color = -1** means draw in the current drawing color. No color attribute is changed.
XgrDrawTextFill **XgrDrawTextFillGrid** **XgrDrawTextFillScaled**	**(grid, color, text$)** **(grid, color, text$)** **(grid, color, text$)** These functions draw text at the appropriate drawpoint after they fill the rectangle that contains the text with the current background color. Then the appropriate drawpoint is left after the last character. **color = -1** means draw in the current drawing color. No color attribute is changed.

XgrFillBox **XgrFillBoxGrid** **XgrFillBoxScaled**	**(grid, color, x1, y1, x2, y2)** **(grid, color, x1Grid, y1Grid, x2Grid, y2Grid)** **(grid, color, x1#, y1#, x2#, y2#)** These functions fill the rectangle with opposite corners at the specified coordinates. **color = -1** means draw in the current drawing color. No color attribute is changed.
XgrGetDrawpoint **XgrGetDrawpointGrid** **XgrGetDrawpointScaled**	**(grid, @x, @y)** **(grid, @xGrid, @yGrid)** **(grid, @x#, @y#)** These functions return the current value of the appropriate drawpoint.
XgrGrabPoint **XgrGrabPointGrid** **XgrGrabPointScaled**	**(grid, @x, @y, @red, @green, @blue, @color)** **(grid, @xGrid, @yGrid, @red, @green, @blue, @color)** **(grid, @x#, @y#, @red, @green, @blue, @color)** These functions grab the color of the pixel at the appropriate drawpoint, and return the drawpoint coordinates and the color of the pixel as 16-bit color intensities **(red,green,blue)**, and a standard color **(color)**. **-1** is returned in **(red,green,blue,color)** if the drawpoint is outside **grid** or the underlying graphics system does not support this operation.
XgrMoveDelta **XgrMoveDeltaGrid** **XgrMoveDeltaScaled**	**(grid, dx, dy)** **(grid, dxGrid, dyGrid)** **(grid, dx#, dy#)** These functions move the appropriate drawpoint to a new position computed by adding the specified offset to its current value.
XgrMoveTo **XgrMoveToGrid** **XgrMoveToScaled**	**(grid, x, y)** **(grid, xGrid, yGrid)** **(grid, x#, y#)** These functions set the appropriate drawpoint to the specified value. These functions are equivalent to **XgrSetDrawpoint()** functions.
XgrSetDrawpoint **XgrSetDrawpointGrid** **XgrSetDrawpointScaled**	**(grid, x, y)** **(grid, xGrid, yGrid)** **(grid, x#, y#)** These functions set the appropriate drawpoint to the specified value. These functions are equivalent to **XgrSetMoveTo()** functions.

Image Functions

```
XgrCopyImage                ( grid, imageGrid )
XgrDrawImage                ( grid, imageGrid, startX, startY, endX, endY )
XgrDrawImageExtend          ( grid, imageGrid, startX, startY, endX, endY )
XgrDrawImageExtendScaled    ( grid, imageGrid, startX, startY, endX, endY )
XgrDrawImageScaled          ( grid, imageGrid, startX, startY, endX, endY )
XgrGetImage                 ( grid, @image[] )
XgrGetImageArrayInfo        (@image[], @bitsPerPixel, @width, @height )
XgrLoadImage                ( fileName$, @image[] )
XgrRefreshGrid              ( grid )
XgrSaveImage                ( fileName$, @image[] )
XgrSetImage                 ( grid, @image[] )
```

XgrCopyImage	**(grid, imageGrid)** **XgrCopyImage()** transfers the image in **imageGrid** into a displayable **grid**, starting with the upper-left corner of **grid** and **imageGrid**. The contents of **imageGrid** is not altered. Drawing outside **grid** is not performed, which clips **imageGrid** if it is larger than **grid**. **XgrCopyImage()** is a relatively fast operation.
XgrDrawImage	**(grid, imageGrid, startX, startY, endX, endY)** **XgrDrawImage()** draws **imageGrid** in **grid** at the drawpoint. **(startX,startY:endX,endY)** are pixel offsets into the image. The left and/or upper portion of an image is skipped if **startX** and/or **startY** is non-zero. The right and/or lower portion of an image is not drawn if **endX** and/or **endY** is reached before the image data is exhausted. If **endX** and/or **endY** is zero, **endX** and/or **endY** are set to the **x** and/or **y** limits of the image. Drawing is not performed outside the grid-box. **XgrDrawImage()** is a relatively fast operation.

XgrDrawImageExtend	**(grid, imageGrid, startX, startY, endX, endY)** **XgrDrawImageExtend()** draws **imageGrid** in **grid** at the drawpoint. **(startX,startY:endX,endY)** are pixel offsets into the image. The left and/or upper portions of an image is skipped if **startX** and/or **startY** is non-zero. If **endX** is reached, or **imageGrid** is exhausted before the grid-box **x** limit is reached, the final pixel is repeated to the grid-box **x** limit. If **endY** is reached, or **imageGrid** is exhausted before the grid-box **y** limit is reached, the final row of pixels is repeated to the grid-box **y** limit. If **endX** and/or **endY** is zero, **endX** and/or **endY** are set to the **x** and/or **y** limits of the image. Drawing is not performed outside the grid-box. **XgrDrawImageExtend()** is a relatively fast operation.
XgrDrawImageExtendScaled	**(grid, imageGrid, startX, startY, endX, endY)** **XgrDrawImageExtendScaled()** draws **imageGrid** in **grid** at the drawpoint. **(startX,startY:endX,endY)** are pixel offsets into the image. The left and/or upper portion of an image is skipped if **startX** and/or **startY** is non-zero. If **endX** is reached, or **imageGrid** is exhausted before the grid-box **x** limit is reached, the final pixel is repeated to the grid-box **x** limit. If **endY** is reached, or **imageGrid** is exhausted before the grid-box **y** limit is reached, the final row of pixels is repeated to the grid-box **y** limit. If **endX** and/or **endY** is zero, **endX** and/or **endY** are set to the **x** and/or **y** limits of the image. Drawing is not performed outside the grid-box. **XgrDrawImageExtendScaled()** interprets image data as scaled coordinates, therefore it will ***zoom*** (enlarge and compress) images of drawings, icons, and video images. **XgrDrawImageExtendScaled()** is a slow operation.

XgrDrawImageScaled	**(grid, imageGrid, startX, startY, endX, endY)** **XgrDrawImageScaled()** draws **imageGrid** in **grid** at the drawpoint. **(startX,startY:endX,endY)** are pixel offsets into the image. The left and/or upper portion of an image is skipped if **startX** and/or **startY** is non-zero. The right and/or lower portion of an image is not drawn if **endX** and/or **endY** is reached before the **imageGrid** is exhausted. If **endX** and/or **endY** is zero, **endX** and/or **endY** are set to the **x** and/or **y** limits of the image. Drawing is not performed outside the grid-box. **XgrDrawImageScaled()** interprets image data as scaled coordinates, therefore it will ***zoom*** (enlarge and compress) images of drawings, icons, and video images. **XgrDrawImageScaled()** is a slow operation.
XgrGetImage	**(grid, @image[])** **XgrGetImage()** returns the **image[]** array from **grid**. This copy of the image in **grid** is in 24-bits per pixel ***DIB*** format, with 8-bits each for **red,green,blue**.
XgrGetImageArrayInfo	**(@image[], @bitsPerPixel, @width, @height)** **XgrGetImageArrayInfo()** returns information about the image in **XLONG** array **image[]**. The contents of **image[]** must be in valid ***DIB*** format. **XgrGetImageArrayInfo()** returns the number of **bitsPerPixel,** and the image **(width,height)** in pixels.

XgrLoadImage	**(fileName$, @image[])** **XgrLoadImage()** loads a graphics image from file **fileName$** into **XLONG** array **image[]**. **fileName$** must contain a valid ***DIB*** format file. After loading, **XgrLoadImage()** converts **image[]** into a 24-bit per pixel, 8-bit per **red,green,blue** ***DIB*** format.
XgrRefreshGrid	**(grid)** **XgrRefreshGrid()** draws an image into a displayable **grid** from the image buffer attached to it. Only the portion of the image buffer that overlaps the grid-box belonging to **grid** is refreshed. Drawing is not performed outside the grid-box or image buffer area.
XgrSaveImage	**(fileName$, @image[])** **XgrSaveImage()** saves **XLONG** array **image[]** in file **fileName$**. It can be loaded with **XgrLoadImage()**, and drawn by **XgrDrawImage()**.
XgrSetImage	**(grid, @image[])** **XgrSetImage()** copies the **image[]** array into **grid**. **image[]** must be in valid ***DIB*** format.

Focus Functions

```
XgrGetMouseInfo                ( window, @grid, @xWin, @yWin, @state, @time )
XgrGetSelectedWindow           ( @window )
XgrSetSelectedWindow           ( window )
```

XgrGetMouseInfo	**(window, @grid, @xWin, @yWin, @state, @time)** **XgrGetMouseInfo()** returns the **grid** that has mouse focus, otherwise the grid that currently contains the mouse cursor. Also returned is the **(xWin,yWin)** position of the mouse cursor in **window,** and the mouse **state**. If the mouse cursor is not in a window and/or grid, then **window** and/or **grid=0**.
XgrGetSelectedWindow	**(@window)** **XgrGetSelectedWindow()** returns the **window** that is currently selected on the default display, which is the window to which keyboard messages are directed. If no window created by the program is currently selected, a **0** is returned in window. Since many programs can have windows on the display at the same time, a value of **0** in **window** is not a rare occurance.
XgrSetSelectedWindow	**(window)** **XgrSetSelectedWindow()** displays **window** if it is hidden or minimized, then selects **window** if it is not already selected. Selecting **window** gives it keyboard focus, so the window argument of subsequent keyboard messages will be **window**.

Message Functions

```
XgrAddMessage                  ( wingrid, message, v0, v1, v2, v3 )
XgrDeleteMessages              ( count )
XgrGetCEO                      (@func )
XgrGetMessages                 (@count, @messages[] )
XgrGetMessageType              ( message, @messageType )
XgrJamMessage                  ( wingrid, message, v0, v1, v2, v3 )
XgrMessageNameToNumber         ( message$, @message )
XgrMessageNames                (@count, messages$[] )
XgrMessageNumberToName         ( message, @message$ )
XgrMessageToMessageType        ( message, messageType )
XgrMessagesPending             (@count )
XgrPeekMessage                 (@wingrid, @message, @v0, @v1, @v2, @v3 )
XgrProcessMessages             ( maxCount )
XgrRedrawWindow                ( window, action, xWin, yWin, width, height )
XgrRegisterMessage             ( message$, @message )
XgrSendMessage                 ( wingrid, message, v0, v1, v2, v3, r0, r1 )
XgrSendMessageToWindow         ( wingrid, message, v0, v1, v2, v3, r0, r1 )
XgrSendStringMessage           ( wingrid, message$, v0, v1, v2, v3, r0, r1 )
XgrSendStringMessageToWindow   ( wingrid, message$, v0, v1, v2, v3, r0, r1 )
XgrSetCEO                      ( func )
```

XgrAddMessage	**(wingrid, message, v0, v1, v2, v3)** **XgrAddMessage()** adds a message to the end of the message queue, composed of **(grid,message,v0,v1,v2,v3)**.
XgrDeleteMessages	**(count)** **XgrDeleteMessages()** deletes **count** messages from the message queue or all messages in the queue, whichever is less.
XgrGetCEO	**(@func)** **XgrGetCEO()** returns the address of the CEO function in **func**. An address of **0** means no CEO function exists. When messages are processed by **XgrProcessMessages()**, they are sent to the **CEO** function before any window functions. See **XgrProcessMessages()** for more detailed information.

XgrGetMessages	**(@count, @messages[])** **XgrGetMessages()** returns the first **count** messages from the message queue in **messages[]** without removing them from the queue. If fewer than **count** messages are waiting in the queue, all are returned and **count** is set to the number of messages returned. **messages[]** must be type **MESSAGE**.
XgrGetMessageType	**(message, @messageType)** **XgrGetMessageType()** returns **messageType=1** if **message** is a window message and **messageType=2** if **message** is a grid message.
XgrJamMessage	**(wingrid, message, v0, v1, v2, v3)** **XgrJamMessage()** jams the message composed of **(wingrid,message,v0,v1,v2,v3)** into the front of the queue, which makes it the next message accessed by **XgrPeekMessages()**, **XgrProcessMessages()**, etc.
XgrMessageNameToNumber	**(messageName$, @messageNumber)** **XgrMessageNameToNumber()** converts a **messageName$** into a **messageNumber**. If **messageName$** has never been registered, **0** is returned in **messageNumber**.
XgrMessageNames	**(@count, @messageNames$[])** **XgrMessageNames()** returns all registered **messageNames$[]** and **count**. The last message name is **messageName$[count]** since **(message=0)** is not a valid message number. The **messageNumber** for each message name is its element number in **messageNames$[]**.
XgrMessageNumberToName	**(messageNumber, @messageName$)** **XgrMessageNumberToName()** returns the **messageName$** corresponding to **messageNumber**. If **messageNumber** has never been assigned, an empty string is returned in **messageName$**.
XgrMessagesPending	**(@count)** **XgrMessagesPending()** returns a **count** of the number of messages in the message queue. **count** will be zero if there are no messages in the message queue.

XgrPeekMessage	**(@wingrid, @message, @v0, @v1, @v2, @v3)** **XgrPeekMessage()** returns the next message in the message queue in **(wingrid,message,v0,v1,v2,v3)** without removing it from the queue. If the message queue is empty, **XgrPeekMessage()** suspends the application until a message becomes available.
XgrProcessMessages	**(count)** **XgrProcessMessages()** executes the next **count** messages in the message queue. If there are no messages in the queue, **XgrProcessMessages()** suspends the program until a message is available, then processes them and returns. If one or more messages is in the queue, **XgrProcessMessages()** processes **count** messages or all pending messages, whichever occurs first, then returns. **count=0** tells **XgrProcessMessages()** to return immediately if no messages are in the message queue, otherwise process one message and return. **count=-1** tells **XgrProcessMessages()** to process all messages. If no messages are in the message queue, **XgrProcessMessages()** suspends the program until a messages becomes available, then processes it and returns. Otherwise **XgrProcessMessages()** processes all messages in the message queue, then returns. **count=-2** tells **XgrProcessMessages()** to process messages as they come in, suspending program execution whenever no messages are in the message queue, and returning only if and when it processes an **ExitMessageLoop** message. First **XgrProcessMessages()** checks to see if a CEO function exists. If it does, it calls the CEO, passing it the message arguments, plus **0** in **r0** and a copy of the window or grid argument in **r1**. The CEO can return **-1** in **r0** to cancel the message. When **XgrProcessMessages()** detects the **-1** in **r0**, it cancels the message and returns without calling any window function it would otherwise call. Then **XgrProcessMessages()** checks to see if a window function exists for the window or grid argument in the message. If it does, it calls the function, passing it the message arguments plus **0** in **r0** and a copy of the window or grid argument in **r1**.

XgrRedrawWindow	**(window, action, xWin, yWin, width, height)** **XgrRedrawWindow()** calls the grid functions assigned to every parentless grid in **window**. It sends each a **Redraw** message including **xWin,yWin,width,height** to define the portion of the window that needs to be redrawn. The grid function can determine from **xWin,yWin,width,height** whether any part of the grid is in this portion of **window** and needs to be redrawn. If the grid function needs to redraw the grid, it sends **Redraw** messages to all kids of grid, if any, passing them the same **xWin,yWin,width,height** arguments.
XgrRegisterMessage	**(message$, @message)** **XgrRegisterMessage()** returns a **message** number to correspond with a **message$** name. If **message$** is already registered, **XgrRegisterEvent()** returns the existing **message** number, otherwise it creates a new one.
XgrSendMessage	**(wingrid, message, v0, v1, v2, v3, r0, r1)** **XgrSendMessage()** calls the window or grid function assigned to the window or grid in **wingrid,** passing it the message arguments it received. First **XgrSendMessage()** examines **message** to determine whether the message is a window message or a grid message. For window messages, **XgrSendMessage()** assumes **wingrid** contains a window number, looks up its window function and calls it, passing it the message arguments. For grid message, **XgrSendMessage()** assumes **wingrid** is a grid number, looks up its grid function and calls it, passing it the message arguments.
XgrSendMessageToWindow	**(wingrid, message, v0, v1, v2, v3, r0, r1)** **XgrSendMessageToWindow()** calls the window function associated with **wingrid,** passing it the arguments it received. First **XgrSendMessageToWindow()** examines **message** to determine whether the message is a window or grid message. For window messages, **XgrSendMessageToWindow()** assumes **wingrid** contains a window number, looks up its window function and calls it, passing it the message arguments. For grid messages, **XgrSendMessageToWindow()** assumes **wingrid** contains a grid number, looks up the window function of the window that contains **wingrid** and calls it, passing it the message arguments.

XgrSendStringMessage	**(wingrid, message$, v0, v1, v2, v3, r0, ANY)** **XgrSendStringMessage()** converts the **message$** string into a **message** number, then calls **XgrSendMessage()**.
XgrSendStringMessageToWindow	**(wingrid, message$, v0, v1, v2, v3, r0, ANY)** **XgrSendStringMessageToWindow()** converts the **message$** string into a **message** number, then calls **XgrSetMessageToWindow()**.
XgrSetCEO	**(func)** **XgrSetCEO()** sets the CEO function to **func,** which makes **func** the active CEO address. **func=0** cancels CEO activity. CEO functions must take exactly **8 XLONG** arguments. See **XgrProcessMessages()** for more information.

Messages

```
MouseDown                ( grid, MouseDown, x, y, state, time, 0, focusGrid )
MouseDrag                ( grid, MouseDrag, x, y, state, time, 0, focusGrid )
MouseEnter               ( grid, MouseEnter, x, y, state, time, 0, focusGrid )
MouseExit                ( grid, MouseExit, x, y, state, time, 0, focusGrid )
MouseMove                ( grid, MouseMove, x, y, state, time, 0, focusGrid )
MouseUp                  ( grid, MouseUp, x, y, state, time, 0, focusGrid )
RedrawGrid               ( grid, RedrawGrid, x, y, state, time, 0, 0 )
TimeOut                  ( grid, TimeOut, 0, 0, 0, 0, 0, 0 )
WindowDeselected         ( window, WindowDeselected, 0, 0, 0, 0, 0, 0 )
WindowDestroyed          ( window, WindowDestroyed, 0, 0, 0, 0 )
WindowDisplayed          ( window, WindowDisplayed, 0, 0, 0, 0, 0, 0 )
WindowHidden             ( window, WindowHidden, 0, 0, 0, 0, 0, 0 )
WindowKeyDown            ( window, WindowKeyDown, x, y, state, time, 0, 0 )
WindowKeyUp              ( window, WindowKeyUp, x, y, state, time, 0, 0 )
WindowMaximized          ( window, WindowMaximized, 0, 0, 0, 0, 0, 0 )
WindowMinimized          ( window, WindowMinimized, 0, 0, 0, 0, 0, 0 )
WindowRedraw             ( window, WindowRedraw, xWin, yWin, width, height, 0, 0 )
WindowResized            ( window, WindowResized, x, y, width, height, 0, 0 )
WindowSelected           ( window, WindowSelected, 0, 0, 0, 0, 0, 0 )
```

MouseDown

```
( grid, MouseDown, x, y, state, time, 0, focusGrid )
```

A **MouseDown** message is added to the message queue when the system detects a mouse button down event has occured.

grid contains the grid number of the grid the mouse cursor was in when the mouse button down event was detected.

x,y contain the local coordinates of the mouse cursor in **focusGrid** when the mouse event was detected.

state contains the button number that was depressed, plus the up/down state of all mouse buttons when the mouse button down event was detected.

time contains the millisecond system time at which the mouse button down event was detected.

MouseDrag

```
( grid, MouseDrag, x, y, state, time, 0, focusGrid )
```

A **MouseDrag** message is added to the queue when the system detects mouse movement while one or more mouse buttons is down.

grid contains the grid number of the grid the mouse cursor was in when the mouse drag event was detected.

x,y contain the local coordinates of the mouse cursor in **focusGrid** when the mouse event was detected.

state contains the up/down state of all mouse buttons when the mouse drag event was detected.

time contains the millisecond system time at which the mouse drag event was detected.

MouseEnter	**(grid, MouseEnter, x, y, state, time, 0, focusGrid)** A **MouseEnter** message is added to the message queue when the system detects mouse movement has made the mouse cursor enter a grid-box. If the mouse movement also made the cursor exit another grid, a **MouseExit** message is added to the message queue before the **MouseEnter** message. **grid** contains the grid number of the grid-box the mouse cursor entered. **x,y** contain the local coordinates of the mouse cursor in **focusGrid** when the mouse event was detected. **state** contains the up/down state of all mouse buttons when the mouse enter event was detected. **time** contains the millisecond system time at which the mouse enter event was detected.
MouseExit	**(grid, MouseExit, x, y, state, time, 0, focusGrid)** A **MouseExit** message is added to the message queue when the system detects mouse movement has made the mouse cursor exit a grid-box. If the mouse movement also made the cursor enter another grid, the **MouseExit** message is added to the message queue before the **MouseEnter** message. **grid** contains the grid number of the grid-box the mouse cursor exited. **x,y** contain the local coordinates of the mouse cursor in **focusGrid** when the mouse event was detected. **state** contains the up/down state of all mouse buttons when the mouse exit event was detected. **time** contains the millisecond system time at which the mouse exit event was detected.

MouseMove	**(grid, MouseMove, x, y, state, time, 0, focusGrid)** A **MouseMove** message is added to the message queue when the system detects mouse movement while no mouse buttons are down. **grid** contains the grid number of the grid the mouse cursor was in when the mouse move event was detected. **x,y** contain the local coordinates of the mouse cursor in **focusGrid** when the mouse event was detected. **state** contains the up/down state of all mouse buttons when the mouse move event was detected. **time** contains the millisecond system time at which the mouse move event was detected.
MouseUp	**(grid, MouseUp, x, y, state, time, 0, focusGrid)** A **MouseUp** message is added to the message queue when the system detects a mouse button up event has occured. **grid** contains the grid number of the grid the mouse cursor was in when the mouse button up event was detected. **x,y** contain the local coordinates of the mouse cursor in **focusGrid** when the mouse event was detected. **state** contains the button number that was depressed, plus the up/down state of all mouse buttons when the mouse button up event was detected. **time** contains the millisecond system time at which the mouse button up event was detected.

RedrawGrid	`( grid, Redraw, x, y, width, height, 0, 0 )` A **RedrawGrid** message is added to the message queue when all or part of a grid is exposed or otherwise becomes visible. **XgrRedrawWindow (window, action, xWin, yWin, width, height)** queues or sends a **RedrawGrid** message to every grid in **window** that is partially or wholly within the specified rectangle, or all grids in **window** if **width <= 0** and/or **height <= 0**. **(x,y,width,height)** are the local coordinates of the smallest rectangle in **grid** that includes all of the exposed area. If **width<=0** and/or **height<=0** the entire grid must be redrawn because the entire grid needs redrawing. Most programs have no provision to partially redraw grids and ignore **x,y,width,height**.
TimeOut	`( grid, TimeOut, 0, 0, 0, 0, 0, 0 )` A **TimeOut** message is added to the message queue when a timer created by **XgrStartGridTimer()** times out. Grid timers are destroyed when they time out, so grids that want periodic **TimeOut** messages must call **XgrStartGridTimer()** every time they receive a **TimeOut** message.
WindowDeselected	`( window, WindowDeselected, 0, 0, 0, 0, 0, 0 )` A **WindowDeselected** message is added to the message queue when a window is deselected. Windows usually stay selected until another is selected. A window is selected when a user points at it with the mouse cursor and presses a mouse button, and when programs call **XgrDisplayWindow()** or **XgrSetSelectedWindow()**. Selecting a window deselects the currently selected window, because only one window can be selected at a time. Selected windows are recognizable on the display because their title-bar and frame are emphasized, usually by a colorful background color. The title-bar and frame of deselected windows are deemphasized. Selecting a window means giving it ***keyboard focus.*** Whenever *GraphicsDesigner* detects a keyboard event, it prepares a keyboard message for the selected window.
WindowDestroyed	`( window, WindowDestroyed, 0, 0, 0, 0, 0, 0 )` A **WindowDestroyed** message is added to the message queue when a window is destroyed, whether by user action or as a result of **XgrDestroyWindow()**.

WindowDisplayed	**(window, WindowDisplayed, 0, 0, 0, 0, 0, 0)** A **WindowDisplayed** message is added to the message queue when a window is displayed for the first time or after being hidden or minimized. Windows are displayed when users click on their icons, and when programs call **XgrDisplayWindow()**.
WindowHidden	**(window, WindowHidden, 0, 0, 0, 0, 0, 0)** A **WindowHidden** message is added to the message queue when a window is hidden from view or minimized. Windows are hidden when programs call **XgrHideWindow()**.
WindowKeyDown	**(window, WindowKeyDown, x, y, state, time, 0, focusGrid)** A **WindowKeyDown** message is added to the message queue when the system detects a keyboard key down event. **window** contains the window number of the currently selected window, which is the window with keyboard focus. **x,y** contain the position of the mouse cursor in the local coordinates of **focusGrid** when the **WindowKeyDown** event was detected. **state** identifies the depressed key and gives the state of the keyboard mode-keys (Alt-Control-Shift) when the **WindowKeyDown** event was detected. **time** contains the value of the system millisecond time when the **WindowKeyDown** event was detected. In some programs, individual grids need to receive key down messages. Furthermore, windows can contain several grids capable of accepting keyboard focus. In these cases, window functions keep track of the grid that most recently had keyboard focus in each window and send it a **KeyDown** message when it receives **WindowKeyDown**. The **WindowKeyDown** message must not be passed to the grid function, because **WindowKeyDown** is a window message and window messages must not be sent to grid functions.

WindowKeyUp	**(window, WindowKeyUp, x, y, state, time, 0, focusGrid)** A **WindowKeyUp** message is added to the message queue when the system detects a keyboard key up event. **window** contains the window number of the currently selected window, which means the window with keyboard focus. **x,y** contain the position of the mouse cursor in the local coordinates of **focusGrid** when the **WindowKeyDown** event was detected. **state** identifies the released key and gives the state of the keyboard mode-keys (Alt-Control-Shift) when the **WindowKeyUp** event was detected. **time** contains the value of the system millisecond time when the **WindowKeyUp** event was detected. In some programs, individual grids need to receive key down messages. Furthermore, windows can contain several grids capable of accepting keyboard focus. In these cases, window functions keep track of the grid that most recently had keyboard focus in each window and send it a **KeyUp** message when it receives **WindowKeyUp**.
WindowMaximized	**(window, WindowMaximized, 0, 0, 0, 0, 0, 0)** A **WindowMaximized** message is added to the message queue whenever a window is maximized. Windows are maximized when users click on the maximize button in the window frame, and when programs call **XgrMaximizeWindow()**.
WindowMinimized	**(window, WindowMinimized, 0, 0, 0, 0, 0, 0)** A **WindowMinimized** message is added to the message queue when a window is minimized. Windows are minimized when users click on the minimize button in the window frame, and when programs call **XgrMinimizeWindow()**.
WindowRedraw	**(window, WindowRedraw, xWin, yWin, width, height, 0, 0)** A **WindowRedraw** message is added to the message queue when all or part of **window** is uncovered, displayed for the first time, or displayed after being hidden or minimized. **xWin,yWin,width,height** defines the smallest rectangle that contains all portions of the window that have been exposed and need to be redrawn to restore the visible part of **window**. If **width<=0** or **height<=0**, all grids in **window** need redrawing.
WindowResized	**(window, WindowResized, xDisp, yDisp, width, height, 0, 0)**

	A **WindowResized** message is added to the message queue when the position and/or size of a window is changed. Users drag window resize grips, and programs call **XgrSetWindowPositionAndSize()**. **xDisp,yDisp** contain the display coordinates of the window, while **width,height** contain the width and height of the window in pixels.
WindowSelected	**(window, WindowSelected, 0, 0, 0, 0, 0, 0)** A **WindowSelected** message is added to the message queue when a window is selected. Windows stay selected until another is selected. Users select a window when they point at it with the mouse cursor and press a mouse button. Programs select a window by calling **XgrSetSelectedWindow()** or **XgrDisplayWindow()**. Selecting a window deselects the currently selected window, because only one window can be selected at a time. Selected windows are recognizable on the display because their title-bar and frame are emphasized, usually by a colorful background color. The title-bar and frame of deselected windows are deemphasized. To select a window is to give it *keyboard focus.* Whenever *GraphicsDesigner* detects a keyboard event, adds a keyboard message for the selected window to the message queue.