SUN/130.5

Starlink Project
Starlink User Note 130.5

David Terrett
Nicholas Eaton

7 July 1995

GWM — X Graphics Window Manager
Version 1.5-1
Programmers' Manual

Abstract

This note describes how to create graphics windows on an X display that do not disappear when a program terminates.

Contents

1 Introduction

X was designed to support "graphical user interfaces" (GUIs) and there are two properties of X that make an X window fundamentally different from a traditional graphics device. Firstly, windows "belong" to applications programs and are deleted when the application exits unlike the picture on an image display which remains there until replaced by something else.

Secondly, applications programs are expected respond to "events" occurring in their windows; things like key presses, mouse movements, etc. One event type that all applications must handle is the "window expose" event which occurs whenever part of a window that was invisible—because another application's window was obscuring it for example—becomes visible. The application is responsible for restoring the contents of the newly exposed part of the window but only an application designed as an X application can do this; an application that is using X via a graphics package such as GKS (SUN/83) or IDI (SUN/65) but is otherwise a conventional application that knows nothing of X, cannot.

The X graphics window manager (not to be confused with the Window Manager which allows you to move windows around the screen, iconize them etc.) makes a window on an X windows display behave like a traditional graphics device by making the lifetime of the window independent of any applications program and by handling window expose events. Applications still send plotting commands directly to the window; they don't have to go via a "server" process, so there is no adverse impact on performance. All communication between the window manager and the application is via the X server and the graphics window manager does not have to run on the same machine as the application. Indeed the manager and applications can even be running on different operating systems.

2 Creating a window

The simplest way to create a window (in terms of understanding what is happening) is to use the `xmake` command and specify the properties that you want the window to have. There are other, more convenient, ways to create windows but they are easier to explain after describing the `xmake` command.

```
xmake name options
```

creates a window with the name *name*. Case matters in window names so `mywindow`, `MYWINDOW` and `MyWindow` are all different.

The following options, which can be abbreviated, are recognized:

`-background colour` The background colour of the window; *colour* is the name of a colour recognized by X windows. Examples:

```
xmake test -background black
xmake test -background yellow
```

The default is `Black`.

`-bg` *`colours`* The same as `-background`.

`-borderwidth` Specifies the width of the window border in pixels. The border width is ignored by many window managers but the value specified is used to calculate the window position so should match the actual border width used by the window manager. The default is 10 which is what the Motif window manager uses.

If the window appears to be incorrectly position when a geometry specification of `-geometry -0-0` is used then changing the border width may correct the problem.

`-colours` *`number`* The number of colour table entries to be allocated to the window; *number* should be an integer greater than or equal to 2.

On a server with a writable colour table, the requested number of colour table entries are reserved for the exclusive use of the window; if there are insufficient free entries in the default colour map, a private colour map is created and the entries allocated from that. On a server with a fixed colour table, entries can be shared between windows and any number up to the maximum supported by the server can be requested.

Examples:

```
xmake test -colours 2
xmake test -col 127
```

The default depends on the properties of the X server.

`-fg` *`colours`* The same as `-foreground`.

`-foreground` *`colour`* The foreground colour of the window; *colour* is the name of a colour or RGB specification in X windows format. Case does not matter but colour names cannot be abbreviated. Examples:

```
xmake test -foreground white
xmake test -foreground SpringGreen
xmake test -foreground #FF00FF}
```

The default is `White`.

`-geometry` *`specification`* The size and position of the window; *specification* is a window geometry specification of the form:

`width``x`*`height`*`+/-`*`xorigin`*`+/-`*`yorigin`*

(`x` is the letter "x" and `+/-` is either a plus sign or a minus sign.) All dimensions are in pixels and the origin is the position of the top left hand corner of the window relative to the top left hand corner of the screen if the origin specification is a positive number, or the position of the bottom right corner of the window relative to the bottom right of the screen if it is negative. All the components of the geometry specification are optional. Examples:

```
xmake test -geometry 780x512
xmake test -geom 600x400+300+300
xmake test -geom +512+368
```

To position the window in the bottom right corner of the screen:

```
xmake test -geom -0-0
```

To position the window in the top right corner:

```
xmake test -geom -0+0
```

The default is `780x512` (this leaves it up to the window manager to decide where to place the window).

The geometry specification is only a hint to the window manager and may be ignored.

`-iconic` Create the window as an icon.

`-interactive` Allows the window size to be set interactively. When the window appears on the screen its size can be modified before the window creation is completed and the window size fixed. Resize the window with the facilities provided by the window manager that you are using and then click the mouse on the interior of the window to indicate that the desired size has been set.

This option is ignored if `-iconic` is specified.

`-nointeractive` The opposite of `-interactive`. This is the default.

`-nooverlay` Do not allocate an overlay plane. This option would be required if a default overlay colour had been set using the "X defaults" file (see below) but a window without an overlay plane was required.

`-ovcolour` *`colour`* Set the colour of the overlay plane. If not specified the overlay plane is set to the foreground colour. Example:

```
xmake test -ovcolour Green
```

`-overlay` Allocate an overlay plane There is no default for this option. The default is not to allocate an overlay plane. Note that overlay planes effectively consume half the available colours on the X server so an overlay should be specified only when required.

`-title` *`title`* The window title that is displayed in the window's title bar; *title* is a character string. If it contains spaces or other special characters it must be enclosed in quotes Example:

```
xmake test -title Test\_Window
```

The default title is "`GWM Window -` *`window name`*"

There are defaults for all these options but they are not necessarily appropriate. You can provide your own defaults by putting them in your "X defaults" file (`~/.Xdefaults`) and to have different defaults for different window names. If you edit your X defaults file, the changes won't have any effect until they have been loaded into the resource database in your X server. This is normally done by the window manager when you first log in to the server but can be done explicitly with the command:

```
xrdb -merge \$HOME/.Xdefaults
```

Rather than attempting to explain in detail what is a powerful but quite complicated system here is an example that shows how to control the properties of GWM windows. Suppose your X defaults file contains the following:

Table 1: GWM X resources

Resource name	Default
background	Black
borderWidth	10
colours	0
foreground	White
geometry	780x512
overlay	False
overlayColour	
iconic	False
interactive	False
title	

```
Gwm.mono*colours: 2
Gwm*colours: 128
```

then a window created with the name "mono" will have 2 colours allocated to it and windows with any other names will have 128 colours allocated.

When a graphics package creates a window for you this is the only mechanism available for specifying the properties of the window. For example, GKS will create a window if it doesn't already exist and each of the four available workstation types of X windows can be assigned different properties as follows:

```
Gwm*foreground: Black
Gwm*background: White
Gwm.GKS\_3800*colours: 2
Gwm.GKS\_3800*geometry: 512x380
Gwm.GKS\_3801*colours: 8
Gwm.GKS\_3801*geometry: 780x512
Gwm.GKS\_3802*colours: 128
Gwm.GKS\_3802*geometry: 780x512
Gwm.GKS\_3803*colours: 128
Gwm.GKS\_3803*interactive: True
```

This defines workstation type 3800 as being a small window with only two colours, suitable for black and white line plots, 3801 is a bit bigger and has 8 colours suitable for colour line plots, 3802 is the same size but with more colours for image display applications and 3803 is the same but the window size is set interactively.

A complete list of the resources used and their default values are shown in Table 1.

3 Using windows

Programs that use GWM (usually via a graphics package such as GKS or IDI) will normally create a new window as required. However, to plot into an existing window you need to know how the graphics package generates the window name. For example, GKS uses a window with the name `GKS_`*`nnnn`* where *nnnn* is the workstation type (the connection identifier is ignored) or whatever that name translates to if treated as an environment variable.

Consult the documentation for the graphics package in question for full details.

4 Resizing windows

Once a window has been created, any resizing does not alter the size of the plotting area seen by applications programs; all that happens is either part of the plotting area becomes invisible or a blank border appears around the plotting area.

You should avoid resizing a window while an application is attached (i.e. plotting into) the window as the application may not be able to handle the resulting change in the picture position correctly.

5 Deleting windows

Windows are deleted from the display with the command:

```
xdestroy name
```

where *name* is the window name. There are no command options.

Resetting the X server (i.e. by logging out) will also delete all windows and all window manager processes will exit.

6 The FORTRAN interface

This section describes the subroutines that can be called from a FORTRAN program to interface with the Graphics Window Manager. All these routines use the *inherited status strategy* which means that if the `STATUS` argument is not set to the value `SAI__OK` on entry the routine will exit without performing any action. If an error occurs during the execution of a routine the `STATUS` will be set and an error will be reported using the error message service (EMS—SUN/104).

6.1 Summary of GWM calls

GWM_CLOSE(STATUS)
: Close the X client-server connection.

GWM_CRWIN(WNAME, STATUS)
: Create a GWM window.

GWM_DSWIN(WNAME, STATUS)
: Destroy a GWM window.

GWM_EXIST(WNAME, EXISTS, STATUS)
: Inquire if a GWM window of the given name exists.

GWM_GETCI(WNAME, IDIM, INDEXS, NCOLS, STATUS)
: Inquire the number of colours and colour indices.

GWM_OPEN(DISPLY, USEDEF, STATUS)
: Establish the X client server connection.

GWM_WSETC(OPTION, VALUE, STATUS)
: Set a character string window option.

GWM_WSETI(OPTION, VALUE, STATUS)
: Set an integer window option.

GWM_WSETL(OPTION, VALUE, STATUS)
: Set a logical window option.

6.2 Making the X connection

The first thing to do when using the FORTRAN interface is to call the routine `GWM_OPEN` which establishes the connection between the client and the server. In this context the client is the cpu running the program and the server is the cpu displaying the image (X allows these cpu's to be different). The first argument passed to `GWM_OPEN` defines this connection, however most applications need not worry about the details of the connection and can use the default (the option of specifying the connection path is included for completeness). The default connection is established with the following call

```
CALL GWM_OPEN( ' ', .TRUE., STATUS )
```

where the second argument is set to true to indicate that the default connection is to be used.

The connection is ended with a call to `GWM_CLOSE`. These two routines (`GWM_OPEN` and `GWM_CLOSE`) should bracket all other routines called from this interface.

As well as establishing the X connection `GWM_OPEN` also sets up an error handler to deal with any non-fatal X errors. Once this has been set up any such errors are reported using EMS. When `GWM_CLOSE` is called the previously active error handler is re-established.

6.3 Creating a GWM window

The routine GWM_CRWIN is used to create a GWM window on the server. The following example shows how a window with the name 'XWINDOWS' is created (remember GWM window names are case sensitive).

```
CALL GWM_CRWIN( 'XWINDOWS', STATUS )
```

On its own this call will create a window with the default characteristics of shape, colour etc. To select different characteristics calls to the routines GWM_WSETx are made before the call to GWM_CRWIN.

Each required window characteristic is selected with a separate call to one of the GWM_WSETx routines, where 'x' represents the type of the characteristic, 'I' for integer, 'L' for logical and 'C' for character string. The following table lists the characteristics and their types.

Characteristic	Type
BACKGROUND	C
BORDERWIDTH	I
COLOURS	I
FOREGROUND	C
HEIGHT	I
ICONIC	L
INTERACTIVE	L
NOINTERACTIVE	L
NOOVERLAY	L
OVCOLOUR	C
OVERLAY	L
TITLE	C
WIDTH	I
XORIGIN	I
XSIGN	I
YORIGIN	I
YSIGN	I

Most of these characteristics are directly comparable to the options of the xmake command. The only major difference is the -geometry option which here is split into its four components WIDTH, HEIGHT, XORIGIN and YORIGIN. The XSIGN and YSIGN characteristic is used to override the sign of XORIGIN and YORIGIN values; setting XSIGN or YSIGN to any value

forces the corresponding origin value to have the same sign. This is only required when setting one of the origin characteristics to -0 in order to position the window at the bottom or right hand side of the screen.

The following example shows the use of most the options to create a window.

```
*   Set up the size and position of the window
     CALL GWM_WSETI( 'WIDTH', 256, STATUS )
     CALL GWM_WSETI( 'HEIGHT', 192, STATUS )
     CALL GWM_WSETI( 'XORIGIN', 200, STATUS )
     CALL GWM_WSETI( 'YORIGIN', 0, STATUS )
     CALL GWM_WSETI( 'YSIGN', -1, STATUS)

*   Allocate 128 colours to this window
     CALL GWM_WSETI( 'COLOURS', 128, STATUS )

*   Define the foreground and background colours
     CALL GWM_WSETC( 'FOREGROUND', 'White', STATUS )
     CALL GWM_WSETC( 'BACKGROUND', 'DarkSlateGrey', STATUS )

*   Give the window a title
     CALL GWM_WSETC( 'TITLE', 'Small_Window', STATUS )

*   Create the GWM window
     CALL GWM_CRWIN( 'XWINDOWS', STATUS )
```

When defining a logical characteristic, such as the INTERACTIVE option, passing a true value in `GWM_WSETL` will activate the option, whereas passing a false value is equivalent to accepting the default.

Once the window has been created all the options are reset to their default values. If a second window is to be created all the necessary characteristics have to be re-defined before the next call to `GWM_CRWIN`.

A GWM window will remain on the server after the program which created it has terminated unless it is explicitly deleted using `GWM_DSWIN`.

6.4 Inquiries

The routine `GWM_EXIST` can be used to inquire if a GWM window of a given name exists on the server. A logical argument is returned with a value of true signifying that the window exists on the server, and a value of false indicating that it does not.

The routine `GWM_GETCI` returns the number of colours and the list of colour indices (pen numbers) allocated to a GWM window. The colour indices are returned in an array dimensioned by the application. If the array is not large enough to contain all the allocated colour indices then the array is filled up to its maximum size and the remaining indices are not returned. The argument specifying the number of colours does however return the actual number of colours allocated, and not the number of colours returned in the array. Consider making the inquiry for the window created in the previous example:

```
*   Define the array to receive the colour indices
     INTEGER IDIM
```

```
      PARAMETER ( IDIM = 64 )
      INTEGER INDEXS( IDIM ), NCOLS

*   Inquire the colour indices of the GWM window
      CALL GWM_GETCI( 'XWINDOWS', IDIM, INDEXS, NCOLS, STATUS )
```

In this case the number of colours returned (NCOLS) will be 128, but only the first 64 colour indices will be returned in the INDEXS array.

6.5 Linking programs with GWM

Stand alone programs are linked with GWM by including ‘`gwm_link`‘ on the `f77` command line.

Adam programs are linked with GMW by including ‘`gwm_link_adam`‘ on the `alink` command line.

6.6 Subroutine Specifications

GWM_CLOSE
Close the X client-server connection

Description:

Close the X client-server connection established by GWM_OPEN. The standard error handler is restored.

Invocation:

```
CALL GWM_CLOSE( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

GWM_CRWIN
Create a GWM window

Description:

Create a GWM window. A window with the given name is created on the current X display. The window is created with the default characteristics unless they have been set using the GWM_WSETx routines. After the window has been created any set characteristics are reset to their default values.

Invocation:

```
CALL GWM_CRWIN( WNAME, STATUS )
```

Arguments:

WNAME = CHARACTER∗(∗) (Given)

The window name.

STATUS = INTEGER (Given and Returned)

The global status.

GWM_DSWIN
Destroy a GWM window

Description:

Destroy a GWM window. The GWM window having the given name on the current X display is destroyed.

Invocation:

```
CALL GWM_DSWIN( WNAME, STATUS )
```

Arguments:

WNAME = CHARACTER∗(∗) (Given)
The window name.

STATUS = INTEGER (Given and Returned)
The global status.

GWM_EXIST
Inquire if a GWM window of the given name exists

Description:

Inquire if a GWM window of the given name exists. The current X display is searched for a GWM window with the given name. The result of the search is returned in the EXISTS argument.

Invocation:

```
CALL GWM_EXIST( WNAME, EXISTS, STATUS )
```

Arguments:

WNAME = CHARACTER∗(∗) (Given)

The window name.

EXISTS = LOGICAL (Returned)

True if a GWM window of the given name exists, otherwise false.

STATUS = INTEGER (Given and Returned)

The global status.

GWM_GETCI
Inquire the number of colours and the colour indices

Description:

Inquire the number of colours and the colour indices allocated to the given window.

Invocation:

```
CALL GWM_GETCI( WNAME, IDIM, INDEXS, NCOLS, STATUS )
```

Arguments:

WNAME = CHARACTER∗(∗) (Given)
The window name.

IDIM = INTEGER (Given)
The declared size of the INDEXS array.

INDEXS = INTEGER(IDIM) (Returned)
Array containing the colour indices allocated to the window. If the number of allocated colours (NCOLS) is greater than the array size (IDIM) only the first IDIM indices are returned.

NCOLS = INTEGER (Returned)
The number of colours allocated to the window.

STATUS = INTEGER (Given and Returned)
The global status.

GWM_OPEN
Establish the X client-server connection

Description:

Establish the X client-server connection. The display name specifies the node on which the server is running. Most applications will use the default device in which case the logical argument USEDEF should be set to true and the display name is ignored. An error handler is established which reports errors via EMS. This routine has to be called before any of the other GWM FORTRAN interface routines. The connection is terminated by the routine GWM_CLOSE.

Invocation:

```
CALL GWM_OPEN( DISPLY, USEDEF, STATUS )
```

Arguments:

DISPLY = CHARACTER∗(∗) (Given)

Display name specifying the network node name and display number of the workstation. The format of the name will be "hostname:number<.screen>" if the transport mechanism is TCP/IP or "hostname::number<.screen>" if the transport mechanism is DECNET. The "hostname" is the name of the host machine to which the display is physically connected. The "number" is the number of the server on the host machine. A single CPU can have one or more servers which are usually numbered starting with zero. On a multiple-screen workstation the optional "screen" number indicates the screen to use. Examples are "cpu:0", "cpu::0", "cpu:0.1" or "cpu::0.1". If USEDEF is true the display name is ignored.

USEDEF = LOGICAL (Given)

If true use the default display, otherwise use the display specified by the DISPLY argument.

STATUS = INTEGER (Given and Returned)

The global status.

GWM_WSETC
Set a character string window option

Description:

The window options are used to control the characteristics of the GWM window and to override the default values. These must be set before the window is created with GWM_CRWIN. The 'TITLE' option is an example of an character string option.

Invocation:

```
CALL GWM_WSETC( OPTION, VALUE, STATUS )
```

Arguments:

OPTION = CHARACTER*(*) (Given)
The option name.

VALUE = CHARACTER*(*) (Given)
The option value.

STATUS = INTEGER (Given and Returned)
The global status.

GWM_WSETI
Set an integer window option

Description:

The window options are used to control the characteristics of the GWM window and to override the default values. These must be set before the window is created with GWM_CRWIN. The 'COLOURS' option is an example of an integer option.

Invocation:

```
CALL GWM_WSETI( OPTION, VALUE, STATUS )
```

Arguments:

OPTION = CHARACTER*(*) (Given)
The option name.

VALUE = INTEGER (Given)
The option value.

STATUS = INTEGER (Given and Returned)
The global status.

GWM_WSETL
Set a logical window option

Description:

The window options are used to control the characteristics of the GWM window and to override the default values. These must be set before the window is created with GWM_CRWIN. A logical option has two values, true or false. A true value means select the option, a false value is equivalent to accepting the default. The 'INTERACTIVE' option is an example of a logical option.

Invocation:

```
CALL GWM_WSETL( OPTION, VALUE, STATUS )
```

Arguments:

OPTION = CHARACTER∗(∗) (Given)
The option name.

VALUE = LOGICAL (Given)
The option value, true or false.

STATUS = INTEGER (Given and Returned)
The global status.

7 Graphics Driver Interface

This section describes the routines that are used by low level graphics software to interact with GWM windows. A working knowledge of Xlib programming is assumed.

> *The routines described here are for the use of "system" software only and all this information in this section is subject to change without notice. Therefore before making any use of it you should consult the Starlink project staff at RAL.*

These return a status value rather than reporting errors; symbolic names for the status values can be found in `gwm_err.h`. Note that these are *not* EMS error codes.

7.1 Anatomy of a Window

A GWM window consists of the following components:

- A window
- A pixmap for saving the contents of the window
- On an X display with a writable colour table, a set of allocated colour cells.
- Optionally, an allocated bit plane for use as an overlay plane (only supported on pseudo colour displays).

Various items of information are stored as properties of the window so that any applications which knows the id of the window can read and write them. The `GWM_Set`*`Xxxx`* and `GWM_Get`*`Xxxx`* routines can be used to access them. The properties are:

`GWM_background` The background colour of the window (a character string)

`GWM_colour_table` An array of integers that lists the colour cells allocated to the window. The first element of the array defines the background colour.

`GWM_foreground` The foreground colour of the window (a character string)

`GWM_name` The window name (a character string). This enables a window id to be translated into a GWM window name.

`GWM_ov_mask` A bit mask that which plane is the overlay plane. The mask has a 1 in every bit position that should be used when writing to the window and a 0 in the bit position corresponding to the overlay plane. A window with no overlay plane has a mask of all 1s.

`GWM_pixmap` The id of the pixmap.

`GWM_x_offset GMW_y_offset` The position of the top left corner of the pixmap relative to the top left corner of the window. Used by the refresh process when copying the pixmap to the window and allows the contents of the window to be scrolled.

`GWM_x_ov_offset GWM_y_ov_offset` As above but for the overlay plane.

7.2 Drawing in the Window

Everything drawn in the window must also be drawn in the pixmap so that the window is refreshed properly. The easiest way to do this is to draw into the pixmap first and then add the x and y offsets to all the absolute coordinates and then draw into the window. The graphics context must have its plane mask set to the value of the overlay mask property (so as to avoid altering the overlay plane) and the foreground set to one of the numbers in the colour table array. If the drawing operation results in anything being drawn with the background colour then the background should also be set to one of the numbers in the colour table array, usually the first.

If the drawing operation involves a large amount of data it may be more efficient to only draw into the pixmap and then copy the pixmap to the window with `XCopyArea`. The destination position should be the x and y offset values and again, the overlay plane must be protected by the plane mask in the gc.

7.3 Drawing in the Overlay Plane

Drawing in the overlay plane is just the same as drawing into the window except that the plane mask must be set to the complement of the overlay mask property and the foreground and background colours must be all 1s and all 0s respectively[1]. The overlay offset values must be added to the coordinates when drawing into the window.

7.4 The Colour Table

On a display with a writeable colour table, the colour table property is a list of the colour cells allocated to the refresh process and must not be altered.

On a display with a fixed colour table the colour table property will list all possible pixel values—since the colour cells can't be changed they can be used by all windows—and the colour table can be used to map "virtual" pixel values to the real pixel values used to write to the window and may be altered. For example, GKS uses the colour table to map colour indices onto pixel values.

If a window has an overlay plane, the colour of the overlay plane is set by setting the colour cells that correspond to values listed in the colour tables ORed with the bit position of the overlay plane (ie. the complement of the plane mask).

7.5 Scrolling

Scrolling of the window and overlay planes is achieved by setting the values of the offset and overlay offset properties. The refresh process is notified of the changes and updates the destination positions it uses when copying the pixmap. However, it does not update the window when the property values change so the application that changes the offsets must copy the window contents to the new position itself.

[1]Of course in actual fact only the bit that corresponds to the overlay plane matters since the rest are masked by the plane mask but all 1s and all 0s is the easiest value to use

7.6 Examples

Some example programs that illustrate how to perform various operations on a GWM window can be found in the gwm examples directory (`$STARLINK_DIR/share/gwm` on Starlink systems).

7.7 Subroutine Specifications

GWM_CreateWindow
Create a window

Description:

A GWM window is created according to the specification in the argument list and the display id of the X connection and the name of the window returned.

Invocation:

```
status = GWM_CreateWindow( argc, argv, &display, &name);
```

Arguments:

int argc (given)
Count of number of arguments

char *argv[] (given)
xmake arguments

Display *display (returned)
display id

char name[] (returned)
window name

GWM_DestroyWindow
Destroy a window

Description:

The X display is searched for the named window and if found, the window name property is removed from the root window and the window and its associated pixmap destroyed.

Invocation:

```
status = GWM_DestroyWindow( display, name);
```

Arguments:

Display *display (given)
Display id

char name[] (given)
Window name

GWM_FindWindow
Find a window

Description:

The X server is searched for a GWM window with the specified name and the id of the window returned.

Invocation:

```
status = GWM_FindWindow( display, name, &win);
```

Arguments:

Display *display (given)
Display id

char name[] (given)
Window name

Window win (returned)
Window id

GWM_GetBgCol
Get background colour

Description:

The value of the GWM_background property is fetched from the window.

Invocation:

```
status = GWM_GetBgCol(display, win_id, &bg);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

char *bg (returned)
Pointer to background colour specification

GWM_GetColTable
Get window's colour table

Description:

The GWM_colour_table property is fetched from the window and pointer to it and the number of values it contains returned.

Invocation:

```
status = GWM_GetColTable( display, win_id, &table, &size);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

long *table (returned)
Pointer to Colour table

unsigned long size (returned)
Number of colour table entries

GWM_GetFgCol
Get foreground colour

Description:
The value of the GWM_foreground property is fetched from the window.

Invocation:

```
status = GWM_GetFgCol(display, win_id, &fg);
```

Arguments:

Display ∗display (given)
Display id

Window win_id (given)
Window id

char ∗fg (returned)
Pointer to foreground colour specification

GWM_GetOvMask
Get overlay mask

Description:

The GMW_ov_mask property is fetched from the window.

Invocation:

```
status = GWM_GetOvMask( display, win_id, &mask);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

unsigned long mask
Overlay plane mask

GWM_GetOvScroll
Get overlay scroll offset

Description:

The values of the GWM_x_ov_offset and GWM_y_ov_offset properties are fetched from the window.

Invocation:

```
status = GWM_GetOvScroll( display, win_id, &xoffset, &yoffset);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

int xoffset (returned)
Overlay scroll offset in x

int yoffset (returned)
Overlay scroll offset in y

GWM_GetPixmap
Get pixmap id

Description:

The value of the GWM_pixmap property is fetched from the window.

Invocation:

```
status = GWM_GetPixmap( display, win_id, &pix_id);
```

Arguments:

Display *display (given)
Display id

Window win_di (given)
Window id

Pixmap pix_id (returned)
Pixmap id

GWM_GetScroll
Get scroll offset

Description:

The values of the GWM_x_offset and GWM_y_offset properties are fetched from the window.

Invocation:

```
status = GWM_GetScroll(display, win_id, &xoffset, &yoffset);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

int xoffset (returned)
Scroll offset in x

int yoffset (returned)
Scroll offset in y

GWM_SetBgCol
Set background colour property

Description:

The value of the GWM_background property is set on the window.

Invocation:

```
status = GWM_SetBgCol(display, win_id, bg);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

char *bg (given)
Background colour specification

GWM_SetColTable
Set colour table

Description:
The GWM_colour_table window property is replaced with the new array of values.

Invocation:

```
status = GWM_SetColTable( display, win_id, table, size);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

int *table (given)
Pointer to new colour table array

unsigned long size (given)
Number of entries in colour table

GWM_SetFgCol
Set foreground colour property

Description:

The value of the GWM_foreground property is set on the window.

Invocation:

```
status = GWM_SetFgCol(display, win_id, fg);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

char *fg (given)
Foreground colour specification

GWM_SetOvScroll
Set overlay scroll offset

Description:

The values of the GWM_x_ov_offset and GWM_y_ov_offset window properties are replaced by the new values.

Invocation:

```
status = GWM_SetOvScroll( display, win_id, xoffset, yoffset);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

int xoffset (given)
New overlay offset value in x

int yoffset (given)
New overlay offset value in y

GWM_SetPixmap
Set pixmap id

Description:

The value of the GWM_pixmap window property is replaced with the new value.

Invocation:

```
status = GWM_SetPixmap( display, win_id, pix_id);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

Pixmap pix_id (given)
New pixmap id

GWM_SetScroll
Set offset offset

Description:

The GWM_x_offset and GWM_y_offset window properties are replaced by the new values.

Invocation:

```
status = GWM_SetScroll( display, win_id, xoffset, yoffset);
```

Arguments:

Display *display (given)
Display id

Window win_id (given)
Window id

int xoffset (given)
New window scroll offset in x

int yoffset (given)
New window scroll offset in y