Transparent Overlay Windows 5

This chapter presents information on the application programming interface (API) that provides transparent overlay window capabilities in the Solaris environment. The chapter includes information on the following topics:

- How overlay windows differ from standard X windows
- How to create and draw to overlay windows
- How to ensure that applications using the overlay window API are portable to a wide range of devices

What are Transparent Overlay Windows?

The transparent overlay extension allows the creation and manipulation of transparent overlay windows. These windows are X windows that allow the user to see through to the underlying window on a per-pixel basis. No special hardware is needed to create and use transparent overlay windows, as this functionality has been implemented in software. Complex transparent overlay manipulation on simple hardware may be time consuming; however, the X server can make use of special overlay hardware if available and the client chooses the correct visuals. Note that, depending on your hardware and needs, you may have to adapt the client color allocations for transparent overlay windows.

Overlay windows allow applications to display temporary imagery in a display window. Users of an application that provides transparent overlays can annotate an image with text or graphical figures, temporarily highlight certain

portions of the imagery, or animate figures that appear to move against the background of the imagery. When geometry in the overlay is cleared, any underlying graphics does not need to be regenerated.

The tranparent overlay extension allows the client to use standard X requests to draw primitives in *opaque paint*, which is a name for the standard way of drawing, or *transparent paint*, which makes affected pixels invisible. The paint type is associated with a standard X graphics context. Window backgrounds may also be set to transparent paint. Transparent overlay windows obey all regular window rules and operating proceedures. For example, a transparent overlay window can be positioned anywhere in the window stacking order, regardless of what hardware the windows are associated with. This is implemented in software with the Solaris X server multiple plane group (MPG) functionality.

The server's multiple plane group capability allows windows from different parts of the hardware to coexist. Each window is associated with a visual, which in turn is associated with hardware. Although some hardware is physically created such that there is a definite "layering" (for example, windows created in a hardware overlay plane might be expected to always be seen above the regular windows), MPG works around this limitation in software. MPG allows the stacking order of the windows to be unaffected by the physical imitations of the hardware. As a result, stacking is simply the same as in the standard server. If overlay hardware is available and requested, MPG takes care of minimizing the work and increasing preformance.

In general, an overlay is a pixel buffer (either physical or software simulated) into which graphics can be drawn. When the overlay is physical (that is, not simulated in software), erasing the overlay graphics does not damage the underlying graphics. This provides a performance advantage when the underlying graphics is complex and requires much time to repaint. When the overlay is in software, erasing the overlay graphics may generate an `Expose` event.

Basic Characteristics of Overlay Windows

An overlay window is a special class of an X `InputOutput` window into which pixels can be rendered transparently. Handles to overlay windows have the X window type `Window`. Just like standard X windows, overlay windows are drawables, and an overlay window handle can be passed to any Xlib drawing routine that takes a `Drawable`.

Overlay windows have extended the set of graphics context attributes to include an attribute for paint type. With the transparent overlay extension, transparent overlay windows can be rendered to with either opaque or transparent paint.

Overlay Window Paint Type

While standard X `InputOutput` windows and other drawables (such as pixmaps) accept only opaque paint, overlay windows permit pixels to be rendered with *transparent paint*. Valid pixel values painted opaquely obscure pixels in underlying windows. Such pixels have associated color values that are displayed. Pixels rendered transparently have no intrinsic color; they derive their displayed color from the pixels that lie beneath.

Valid pixel values for pixels painted opaquely are obtained via `XAllocColor()` or another standard pixel allocation mechanism. Painting opaquely with a non-valid pixel value, for example a value that falls outside the valid colormap entries for a visual, produces undefined results for both overlay windows and standard X `InputOutput` windows.

Paint type is defined with the data structure `XSolarisOvlPaintType`. By default, the paint type of a `GC` is opaque. The `XSolarisOvlPaintType` data structure is defined as:.

```
typedef enum {
    XSolarisOvlPaintTransparent,
    XSolarisOvlPaintOpaque,
} XSolarisOvlPaintType;
```

Overlay Window Viewability

An overlay window is considered viewable even if all its pixels are fully transparent. For viewable pixels in an overlay window that are fully transparent, the underlying pixels in the underlay will be displayed.

If an overlay window is unmapped or moved, the underlay beneath may receive exposure events. This, for example, is the case on devices that can not display the overlay window and underlay window in different plane groups.

Rendering Transparent Paint

Applications can render into overlay windows using Xlib primitives. In addition, applications can render transparent paint to overlay windows through a Solaris Visual graphics library, such as the XGL graphics library, by specifying in the `GC` for that library that the paint is to be transparent. Each Solaris Visual library has a defined way of rendering into a transparent overlay window. See the library's documentation for information.

More on Overlay Window Characteristics

In most respects, an overlay window is just like a standard X `InputOutput` window. Specifically, an overlay window has these characteristics:

- It can be mapped or unmapped. The routines `XMapWindow`, `XUnmapWindow`, `XMapSubwindows`, and `XUnmapSubwindows` apply.
- An overlay window can possess its own cursor or use its parent's cursor. In other words, `XDefineCursor` and `XUndefineCursor` apply to overlay windows.
- An overlay window appears in the output of `XQueryTree`.
- The `event_mask` and `do_not_propagate_mask` window attributes function normally. An overlay window can express interest in any type of event.
- `XTranslateCoordinates` and `XQueryPointer` apply to overlay windows.
- `save_under` applies as for standard X windows.
- `override_redirect` applies as for standard X windows.

An overlay window also has some characteristics that makes it unique as a window. The following sections describe these characteristics.

Overlay Window Background

As defined in the X specification, windows can have a *background*. The main purpose of window background is to display something in the exposed areas of a window in case the client is slow to repaint these areas. This background is rendered whenever the window receives an `Expose` event. The background is rendered before the `Expose` event is sent to the client. The background is also rendered when the client makes an `XClearArea` or `XClearWindow` request.

Like standard X `InputOutput` windows, overlay windows can also have a background. The background of an overlay window is rendered just like a non-overlay window in response to `Expose` events, `XClearArea` requests, or `XClearWindow` requests. In addition to the standard types of background (None, pixmap, pixel, or parent relative), overlay windows can also be assigned a new type of background: transparent. A new routine, `XSolarisOvlSetWindowTransparent`, is available to set the background type to transparent.

The background of an overlay window is transparent by default. However, the application can still specify one of the usual X types of background: `None`, a pixmap XID, a pixel value, or `ParentRelative`, as shown in Table 5-1.

Table 5-1 Background Values for an Overlay Window

Background	Description
transparent	Background of overlay window is transparent by default.
None	No rendering is performed when the overlay window encounters a condition that invokes background painting. Neither transparent nor opaque paint is rendered.
Pixmap ID	The background is rendered with opaque paint. The rendered pixel values are derived from the pixmap as defined in the X specification.

Table 5-1 Background Values for an Overlay Window

Background	Description
Single pixel value	The background is a solid color rendered with opaque paint.
ParentRelative	The behavior for a ParentRelative background depends on the parent window background and its type. If the parent window is an underlay, the background for the overlay window child will be rendered with opaque paint, and the rendered pixels will be as defined in the X specification. If the parent window is an overlay, the background of the overlay child will be the same as that of the parent, either transparent or opaque paint will be rendered.

Attempts to set the background of a non-overlay window with `XSolarisOvlSetTransparent` generates a `BadMatch` error. If an underlay window has a `ParentRelative` background and the parent window is an overlay with a transparent background, the underlay child is treated as if it has a background of `None`.

Overlay Window Border

The border of overlay windows is opaque. It is always drawn with opaque paint. Just like standard X `InputOutput` windows, the border width can be controlled with `XSetWindowBorderWidth`.

Overlay Window Backing Store

An overlay window can be granted backing store not only for the color information of its opaque pixels, but also for the paint type of its pixels. If the `backing_store` attribute of a window is set to `Always` or `WhenMapped`, the X11 server can grant backing store for an overlay window. When backing store is granted, both the color and paint information are retained.

The `backing_planes` and `backing_pixel` apply only to the color information of opaque pixels in the window.

Overlay Window Gravity

The bit and window gravity attributes (`bit_gravity` and `win_gravity`) apply to overlay windows. However, if the gravity calls for the movement of pixels, the transparency information is moved, along with the pixel color information.

Overlay Colormap

Overlay colormap installation follows the X rules. If your application uses pixel-sharing overlay/underlay pairs, create a single colormap for both windows. Refer to "Choosing Visuals for Overlay/Underlay Windows" on page 65 and "Designing an Application for Portability" on page 83 for more on the subject of pixel-sharing pairs.

If the pair is known never to share hardware color LUTs, different colormaps can be safely assigned to the overlay and underlay window without the occurrence of colormap flashing.

Note – To improve the portability of applications and to minimize color flashing, use colormaps with the same colors in both the overlay and underlay window colormaps. If this is not possible, use one of the visual inquiry routines to determine whether different colormaps can be assigned without producing flashing.

Input Distribution Model

Overlay windows can express interest in events just like a standard X window. An overlay window receives any event that occurs within its visible shape; the paint type of the pixel at which the event occurs doesn't matter. For example, if the window expresses interest in window enter events, when the pointer enters the window's visible shape, the window receives a window enter event, regardless of whether the pixel is opaque or transparent.

This has some implications for how applications should implement interactive *picking* (selection) of graphical objects. Applications that draw graphical figures into an overlay window above other graphical figures drawn into the underlay window should express interest in events in either the overlay or underlay

window, but not both. When the application receives an input event, it must use its knowledge of the overlay/underlay layering to determine which graphical figure has been picked.

For example, let's say the application expresses interest in events on the underlay window. When the application receives an event at coordinate (x, y), it should first determine if there is a graphical figure at that coordinate in the overlay. If so, the search is over. If not, the application should next see if there is a graphical figure at that coordinate in the underlay.

Print Capture

After graphical imagery has been rendered to an X window, the user may want the window contents to be captured and sent to a printer for hard copy output. The most widespread technique for doing this is to perform a *screen dump*, that is, to read back the window pixels with `XGetImage`, and to send the resulting image to the printer. To fit the image to the size of the printed page, some image resampling may be necessary. This can introduce *aliasing* artifacts into the image.

Another print capture technique that is growing in popularity in the X11 community is to re-render the graphics through a special printer graphics API. This API supports the standard Xlib graphics calls. It converts these calls into a page description language (PDL) format and sends it to the appropriate print spooler. The advantage of this technique is that the graphics can be scaled to fit the printed page by scaling the coordinates themselves rather than the pixels after scan conversion has been applied. As a result, aliasing artifacts are minimized.

The print API technique has a significant drawback when applied to an overlay/underlay window pair. Most PDLs only support the notion of opaque paint; they do not provide for the marking of transparent paint. In the PostScript PDL, for example, the marked pixels always supersede what was previously marked. Given such a limitation, it is not always possible to capture the imagery in an overlay/underlay window pair using this technique. Certainly, in applications where the background of the overlay is completely transparent and only opaque paint is drawn to it, the underlay could be marked first and the overlay marked second. But if transparent paint was drawn to the overlay, erasing other opaque paint in the overlay, this would not work.

Until this issue is resolved, capture overlay windows and send them to the printer using `XReadScreen` and resampling. Alternatively, do not use overlays to render information that is to be printed.

Choosing Visuals for Overlay/Underlay Windows

The Solaris transparent overlay API supports multiple plane group (MPG) and single plane group (SPG) devices. Display devices come in a wide variety of configurations. Some have multiple plane groups. Some have multiple hardware color lookup tables (LUTs). Some dedicate color LUTs to particular plane groups and some share color LUTs between plane groups. This wide variety makes it difficult for an application writer to construct portable overlay applications.

For a given type of underlay window, some devices can provide some types of overlay windows with high-performance rendering. Other devices provide the same type of overlay window but with slower rendering. Some devices can support overlays with many colors, and some devices cannot. Some devices can support simultaneous display of both overlay and underlay colors for all types of overlays and underlays. Others support simultaneous display of colors but not for all overlay/underlay combinations. Still others support a certain degree of simultaneous color display. These devices support more than one hardware color LUT. Hardware might not contain enough color LUTs to enable all applications to display their colors simultaneously.

The Solaris Visual Overlay Window API provides two utility routines to enable an application to negotiate with the system for a suitable overlay/underlay visual pair:

- `XSolarisOvlSelectPartner`
- `XSolarisOvlSelectPair`

These routines are described in the section “Designing an Application for Portability” on page 83.

The assumption is made that each application has an ideal configuration of windows and colors. An application should start out by asking for the “best” overlay/underlay pair. If this can be satisfied by the device, then the negotiation is complete, and the application proceeds to create windows on the selected underlay and overlay visuals. But if no visual pair satisfies the query, the application must relax its demands. To this end, it should specify the “next

best" pair. The application may choose to ask for less colorful visuals, or it may accept lower rendering performance on one of the visuals. The process continues until either a satisfactory visual is found, or the application decides it's not worth running in this environment without certain criteria being met.

The overlay API provides routines that enable the application to conduct such a negotiation in a single subroutine call. The application specifies criteria to be matched for either the overlay visual, the underlay visual, or both. Application programmers are encouraged to use these routines to ensure portability to the widest range of graphics devices.

Example Program

The program below demonstates a simple example of a transparent overlay. The program creates a transparent overlay window, draws the window border in white, displays a text string in white, and draws a white filled rectangle. The paint type is opaque by default, and the window background is transparent by default. Use the following `Makefile` to compile and link the program.

```
simple: simple.c
    cc -I../ -I/usr/openwin/include -o simple simple.c \
    -L/usr/openwin/lib -lX11 -lXext
```

Code Example 5-1 Transparent Overlay Example Program

```
#include <stdio.h>
#include <X11/Xlib.h>
#include "X11/Xmd.h"
#include <X11/extensions/transovl.h>
#include <X11/extensions/transovlstr.h>

Display                *display;
Window                 window;
XSetWindowAttributes   attribs;

GC                     gc;
XGCValues              gcvalues;

main()
{
 display = XOpenDisplay("");
 attribs.override_redirect = True;
 attribs.border_pixel = WhitePixel(display, 0);
```

```
window = XSolarisOvlCreateWindow(display,
             DefaultRootWindow(display),
             100, 100, 500, 500, 10,
             CopyFromParent, InputOutput,CopyFromParent,
             CWBorderPixel | CWOverrideRedirect, &attribs);
gcvalues.font = XLoadFont(display, "fixed");
gcvalues.foreground = WhitePixel(display, 0);
gc = XCreateGC(display, window, GCFont | GCForeground, &gcvalues);
XMapWindow(display, window);
XDrawString(display, window, gc, 50, 50, "This is a test", 14);
XFillRectangle(display, window, gc, 70, 70, 100, 100);
XFlush(display);
while (1);
}
```

Overview of the Solaris Transparent Overlay Window API

The transparent overlay window API includes the routines listed in Table 5-2. These routines are provided by `libXext.so`. To use the Solaris overlay routines, do the following:

- Include the file `/usr/openwin/include/X11/extensions/transovl.h`
- Link the library device handler with the library `/usr/openwin/lib/libXext.so`

Table 5-2 List of Overlay Window Routines

Name	Description
`XSolarisOvlCreateWindow`	Creates an overlay window.
`XSolarisOvlIsOverlayWindow`	Indicates whether a window is an overlay window.
`XSolarisOvlSetPaintType`	Specifies the type of paint rendered by subsequent Xlib drawing.
`XSolarisOvlGetPaintType`	Gets the current paint type.
`XSolarisOvlSetWindowTransparent`	Sets the background state of an overlay window to be transparent.

5

Table 5-2 List of Overlay Window Routines

Name	Description
`XSolarisOvlCopyPaintType`	Renders opaque and transparent paint into the destination drawable based on the paint type attributes of the pixels in the source drawable.
`XSolarisOvlCopyAreaAndPaintType`	Copies the area and paint type from one pair of drawables to another.
`XReadScreen`	Returns the displayed colors in a rectangle of the screen.
`XSolarisOvlSelectPartner`	Returns the optimal overlay or underlay visual for an existing visual.
`XSolarisOvlSelectPair`	Selects an optimal overlay/underlay pair that best meets a set of defined criteria for the overlay and underlay visuals.

The remainder of this chapter discusses the transparent overlay API routines.

Creating Overlay Windows

You can create an overlay window using `XSolarisOvlCreateWindow`. This routine behaves exactly as `XCreateWindow` except that the resulting window is an overlay window. The newly created window can be rendered into with both opaque and transparent paint, and the background of the overlay window is transparent.

The `class` argument to `XSolarisOvlCreateWindow` should be `InputOutput`. An overlay window can be created as an `InputOnly` window but, in this case, it will behave like a standard `InputOnly` window. It is only for `InputOutput` windows that there is a difference between overlay and non-overlay.

The syntax and arguments for `XSolarisOvlCreateWindow` are shown below.

```
Window
XSolarisOvlCreateWindow(Display *display, Window parent, int x, int y,
        unsigned int width, unsigned int height,
        unsigned int border_width, int depth, unsigned int class,
        Visual * visual, unsigned long valuemask,
        XSetWindowAttributes * attr)
```

The arguments for this routine are the same as those for `XCreateWindow`.

`display`	Specifies the connection to the X server.
`parent`	Specifies the parent window.
`x, y`	Specifies the coordinates of the upper-left pixel of this window, relative to the parent window.
`width, height`	Specifies the width and height, in pixels, of the window.
`border_width`	Specifies the width, in pixels, of the window's borders.
`depth`	Specifies the depth of the window.
`class`	Specifies the class of the window. If the class is not `InputOutput`, the window will not be an overlay window.
`visual`	Specifies a pointer to the visual structure for this window.
`valuemask`	Specifies which window attributes are defined in the `attr` argument.
`attr`	Specifies the attributes of the window.

You can use any visual to create the overlay. However, not all overlay/underlay visual pairs may be optimal. Each screen defines a set of optimal overlay/underlay visual pairs. These define the optimal visuals of the overlay windows that can be created with a particular underlay visual.

Likewise, they define the optimal visuals of underlay windows that can be created with a particular overlay visual. You can determine the optimal pairs using `XSolarisOvlSelectPair` and `XSolarisOvlSelectPartner`.

The definition of *optimal* varies from device to device, but it will usually refer to the ability of a device to create an overlay window in a different plane group than that of an underlay window. See page 89 for more information on overlay/underlay visual pairs.

Overlay windows are destroyed with the Xlib routines `XDestroyWindow` or `XDestroySubwindows`.

Setting the Paint Type of a Graphics Context

You can set a `GC`'s paint type with the `XSolarisOvlSetPaintType` routine. `XSolarisOvlSetPaintType` specifies the type of paint rendered by subsequent Xlib drawing with the given `GC`. It controls whether Xlib drawing routines using this `GC` produce opaque or transparent pixels on overlay windows. The paint type specified applies to the `GC` until it is changed by another call to this routine. The paint type attribute applies to both the foreground and background `GC` attributes. The syntax and arguments are shown below.

```
void
XSolarisOvlSetPaintType (Display *display, GC gc,
       XSolarisOvlPaintType paintType)
```

`display`	Specifies the connection to the X server.
`gc`	Specifies the affected `GC`.
`paintType`	Specifies the type of paint rendered by subsequent Xlib drawing routines using the specified `GC`.

The value of `paintType` can be `XSolarisOvlPaintOpaque` or `XSolarisOvlPaintTransparent`.

- If the value of `paintType` is `XSolarisOvlPaintOpaque`, the pixels generated by subsequent Xlib drawing routines with this `GC` will be opaque. This means the pixels will obscure underlying pixels. This is the default.

- If the value of `paintType` is `XSolarisOvlPaintTransparent`, the pixels generated by subsequent Xlib drawing routines with this `GC` will be transparent. This means that, for these pixels, the color of the underlying pixels is displayed.

Setting the Background State of an Overlay Window

You can set the background state of an overlay window to be transparent with the `XSolarisOvlSetWindowTransparent` routine. Any background rendering that occurs after this request causes the background to be transparent. To change background state to any other value, use `XChangeWindowAttributes()`, `XSetWindowBackground()`, or `XSetWindowBackgroundPixmap()`.

The syntax and arguments of `XSolarisOvlSetWindowTransparent` are shown below.

```
void
XSolarisOvlSetWindowTransparent (Display *display, Window w)
```

`display`	Specifies the connection to the X server.
`w`	The overlay window.

Note – If `w` is not an overlay window, a `BadMatch` error results.

Rendering to an Overlay Window

Once an overlay window is created, you can use all the standard Xlib primitive rendering routines, such as `XDrawLines` and `XFillRectangles`, to draw into the window. When drawing to overlay windows, the paint type attribute of the `GC` is used to control the quality of the pixels rendered. The paint type attribute applies to both the foreground and background `GC` attributes. To set the paint type, use the `XSolarisOvlSetPaintType` routine; for information on this routine, see page 70.

The paint type of the GC also controls the type of pixels rendered with XPutImage. If the paint type of the argument GC is XSolarisOvlPaintOpaque, the color information from the source image is used and the pixels are rendered with opaque paint. However, if the paint type is XSolarisOvlPaintTransparent, the source color information is ignored, and the pixels are rendered with transparent paint.

If a GC with a paint type of XSolarisOvlPaintTransparent is used to render to a drawable other than an overlay window, such as an underlay window or pixmap, the GC paint type is ignored, and the pixels are rendered with opaque paint.

Querying the Characteristics of an Overlay Window

You can determine whether a window is an overlay window using the routine XSolarisOvlIsOverlayWindow. You can also determine a GC's current paint type using the routine XSolarisOvlGetPaintType.

Determining Whether a Window is an Overlay Window

You can use the routine XSolarisOvlIsOverlayWindow to determine whether a window is an overlay window. The routine returns True if the given window w is an overlay window and returns False otherwise.

```
Bool XSolarisOvlIsOverlayWindow (Display *display, Window w)
```

display	Specifies the connection to the X server.
w	Specifies the window.

Determining the Paint Type of a Graphics Context

The routine XSolarisOvlGetPaintType returns the GC's current paint type.

```
XSolarisOvlPaintType
XSolarisOvlGetPaintType (Display *display, GC gc)
```

`display`	Specifies the connection to the X server.
`gc`	The `GC` to be inquired about.

Pixel Transfer Routines

The Solaris overlay API provides three pixel transfer routines:

- `XSolarisOvlCopyPaintType` – Renders opaque and transparent point into a destination drawable based on the paint type attributes of the source drawable.
- `XSolarisCopyAreaAndPaintType` – Copies an area and its paint type from one pair of drawables to another.
- `XReadScreen` – Returns the colors displayed in a given area of the screen.

The existing Xlib pixel transfer routines `XGetImage`, `XCopyArea`, and `XCopyPlane` can also be used with overlay windows. The use of these routines is described below.

Filling an Area Using the Source Area Paint Type

The `XSolarisOvlCopyPaintType` routine uses the paint type information of a specified rectangle in a source rectangle to control a fill operation in a specified rectangle in a destination rectangle. The source rectangle and destination rectangle can be any type of drawable. If the source rectangle is an overlay, the paint type attribute of its pixels is used as the source of the copy, and the color information is ignored. If the source rectangle is any other type of drawable, the bit plane specified in the routine is treated as if it were paint type data and it is used for the copy. In this case, the bit plane must have only one bit set.

The syntax and arguments are shown below.

```
void
XSolarisOvlCopyPaintType(Display *display, Drawable src,
       Drawable dst, GC gc, int src_x, int src_y,
       unsigned int width, unsigned int height, int dest_x,
       int dest_y, unsigned long action, unsigned long plane)
```

`display`	Specifies the connection to the X server.
`src`	Specifies the source drawable from which to obtain the paint type information.
`dst`	Specifies the destination drawable.
`gc`	Specifies the `GC`.
`src_x`, `src_y`	Specify the x and y coordinates of the upper-left corner of the source rectangle relative to the origin of the source drawable.
`width`, `height`	Specify the width and height of both the source and destination rectangles.
`dest_x`, `dest_y`	Specify the x and y coordinates of the upper-left corner of the destination rectangle relative to the origin of the destination drawable.
`action`	Specifies which paint type data is to be copied. This can be one of `XSolarisOvlCopyOpaque`, `XSolarisOvlCopyTransparent`, or `XSolarisOvlCopyAll`.
`plane`	Specifies the bit-plane of the `src` drawable to be used as paint type information when the source is not an overlay.

`src` and `dst` must have the same screen, or a `BadMatch` error results.

Table 5-3 summarizes the possible combinations of `src` and `dst` and their actions. The left side of the table shows the possible `src` combinations. The top of the table shows the possible `dst` combinations. The actions A1-A4 are explained following the table.

Table 5-3 XSolarisOvlCopyPaintType Source/Destination Combinations and Actions

Source/Destination	Overlay	Drawable
overlay	A1	A2
drawable	A3	A4

- A1—Opaque pixels in the source overlay cause the corresponding pixels in the destination to be filled with opaque color as specified by the fill attributes of the `GC`. Transparent pixels in the source cause the corresponding pixels in the destination to be filled with transparent paint.
- A2—Opaque pixels in the source overlay cause the corresponding pixels in the destination to be filled according to the fill attributes of the `GC`. Transparent pixels in the source overlay cause the corresponding pixels in the destination to be filled according to the same fill attributes of the `GC`, but with the foreground and background pixels swapped.
- A3—The pixels in the destination overlay are filled with opaque paint or made transparent as in A1 above depending on the bit values of the source drawable's `plane`. Bit values of 1 in the source are treated as if they were opaque pixels and bit values of 0 are treated as if they were transparent.
- A4—The pixels in the destination drawable are filled with paint as in A2 above depending on the bit values of the source drawable's plane. Bit values of 1 in the source bit plane are treated as if they were opaque pixels and bit values of 0 are treated as if they were transparent.

The `action` argument specifies whether opaque paint (`XSolarisOvlCopyOpaque`), transparent paint (`XSolarisOvlCopyTransparent`), or both (`XSolarisOvlCopyAll`) should be operated upon. This allows a client to *accumulate* opaque or transparent paint.

If portions of the source rectangle are obscured or are outside the boundaries of the source drawable, the server generates `Expose` events, using the same semantics as `XCopyArea`.

This routine uses these GC components: function, plane-mask, fill-style, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask. It might use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin.

XSolarisOvlCopyPaintType can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

Copying an Area and Its Paint Type

The XSolarisCopyAreaAndPaintType routine copies the specified area of source drawable for the color information to the specified area of destination drawable for color information. If the destination drawable is not an overlay, it also fills the specified areas of paint type information destination drawable according to the paint type information specified in the paint type information source drawable.

You can use XSolarisOvlCopyAreaAndPaintType to combine an image in the client's memory space (consisting of color and/or paint type information) with a rectangle of the specified overlay window. To do this, first move the image and paint type data into the server: use XPutImage to copy the data into two pixmaps of the appropriate depths. Then call XSolarisOvlCopyAreaAndPaintType with the color and paint type drawables to copy information to the overlay.

You can also use XSolarisOvlCopyAreaAndPaintType to retrieve pixel information (color and/or paint type information) from a specified drawable. To do this, call XSolarisOvlCopyAreaAndPaintType with two separable destination drawables. To get the data from the server into the client's memory space, call XGetImage on each of the drawables.

The syntax and arguments for `XSolarisCopyAreaAndPaintType` are shown below.

```
void
XSolarisOvlCopyAreaAndPaintType(Display * display, Drawable
colorsrc,
       Drawable painttypesrc, Drawable colordst,
       Drawable painttypedst, GC colorgc, GC painttypegc,
       int colorsrc_x, int colorsrc_y, int painttypesrc_x,
       int painttypesrc_y, unsigned int width, unsigned int height,
       int colordst_x, int colordst_y, int painttypedst_x,
       int painttypedst_y, unsigned long action, unsigned long plane)
```

`display`	Specifies the connection to the X server.
`colorsrc`	The color information source drawable. `colorsrc` can be any depth drawable or an overlay window.
`painttypesrc`	The paint type information source drawable. `painttypesrc` can be any drawable or an overlay window. If `painttypesrc` is not an overlay window, the bit plane of `painttypesrc` specified in `plane` is treated as if it were paint type data and it is used for the copy. `plane` must have only one bit set in this case.
`colordst`	The color information destination drawable.
`painttypedst`	The paint type information destination drawable. If `colordst` is an overlay, this drawable will be ignored.
`colorgc`	The `GC` to use for the color information copy.
`painttypegc`	The `GC` to use to fill areas in `painttypedst`. If `colordst`/`painttypedst` is an overlay, this `GC` will be ignored.
`colorsrc_x` `colorsrc_y`	The X and Y coordinates of the upper-left corner of the source rectangle for color information relative to the origin of the color source drawable.

`painttypesrc_x` `painttypesrc_y`	The X and Y coordinates of the upper-left corner of the source rectangle for paint type information relative to the origin of the paint type source drawable.
`width, height`	The dimensions in pixels of all the source and destination rectangles.
`colordst_x` `colordst_y`	The X and Y coordinates of the upper-left corner of the destination rectangle for color information relative to the origin of the color destination drawable.
`painttypedst_x` `painttypedst_y`	The X and Y coordinates of the upper-left corner of the destination rectangle for paint type information relative to the origin of the paint type destination drawable. If `colordst/painttypedst` is an overlay, colordst_x and colordst_y will be used.
`action`	Specifies which paint type data is to be copied. This can be one of XSolarisOvlCopyOpaque, XSolarisOvlCopyTransparent, or XSolarisOvlCopyAll.
`plane`	Specifies the source bit-plane in `painttypesrc` to be used as paint type information when `painttypesrc` is not an overlay.

`colordst` can be any drawable, but must be of the same depth and have the same root as `colorsrc`, otherwise, a `BadMatch` error results. If `colordst` is an overlay, then `painttypedst` is ignored, otherwise `painttypedst` can be any type of drawable.

Table 5-4 summarizes the possible combinations of sources and destinations and their respective actions. The left side of the table shows the possible `colorsrc/painttypesrc` combinations and the top of the table shows the possible `colordst/painttypedst` combinations. The actions A1-A8 are

explained below the table. An Impossible entry in the table indicates that the given combination is impossible, since the `painttypedst` is ignored when the `colordst` is an overlay.

Table 5-4 XSolarisOvlCopyAreaAndPaintType Source/Destination Combinations and Actions

	Overlay/Overlay	**Overlay/Drawable**	**Drawable/Overlay**	**Drawable/Drawable**
overlay/overlay	A1	Impossible	A5	A5
overlay/drawable	A2	Impossible	A6	A6
drawable/overlay	A3	Impossible	A7	A7
drawable/drawable	A4	Impossible	A8	A8

- A1—The paint type information from `painttypesrc` is used as a mask to copy the color information from `colorsrc` to `colordst`. Opaque pixels in `painttypesrc` cause the corresponding pixel in `colorsrc` to be copied to `colordst`, transparent pixels cause the corresponding pixel in `colordst` to be made transparent. If a transparent pixel from `colorsrc` is copied to `colordst`, the actual color transferred will be undefined.
- A2—Same as A1 except that the paint type information is extracted from the bit-`plane` of `painttypesrc` specified by `plane`. A bit value of 1 indicates an opaque pixel whereas a bit value of 0 indicates transparent.
- A3—Same as A1 except that a non-overlay drawable is used to obtain the color information so there will be no undefined colors due to transparent pixels.
- A4—Same as A3 except that the paint type information is taken from the specified bit-plane of `painttypesrc` as in A2.
- A5—The paint type information from `painttypesrc` is used as a mask to copy the color information from `colorsrc` to `colordst` as in A1. In addition, the paint type information controls rendering to the `painttypedst` drawable as in `XSolarisOvlCopyPaintType`.
- A6—Same as A5 except that the paint type information is taken from the specified bit-plane of `painttypesrc` as in A2.
- A7—Same as A5 except that there will be no undefined colors due to transparent color source pixels.
- A8—Same as A7 except that the paint type information is taken from the specified bit-plane of `painttypesrc` as in A2.

The action argument specifies whether opaque paint (XSolarisOvlCopyOpaque), transparent paint (XSolarisOvlCopyTransparent), or both (XSolarisOvlCopyAll) should be copied. This allows a client to accumulate opaque or transparent paint.

NoExpose and GraphicsExpose events are generated in the same manner as XSolarisOvlCopyPaintType.

If an overlay is used for the colordst argument, the painttypedst, painttypegc, painttypedst_x and painttypedst_y arguments will all be ignored. A NULL pointer can be used for painttypegc and a value of None can be used for painttypedst. The overlay will have the exact paint type defined by the pixels in the area specified in painttypesrc. The color information copy will not affect the destination paint type.

This function uses these GC components from colorgc: function, plane-mask, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask.

If colordst is not an overlay then this function will use these GC components from painttypegc: function, plane-mask, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. In addition, it may also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XSolarisOvlCopyAreaAndPaintType can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

Retrieving Overlay Color Information

The routine XReadScreen returns the displayed colors in a rectangle of the screen. It thus provides access to the colors displayed on the screen of the given window.

On some types of advanced display devices, the displayed colors can be a composite of the data contained in several different frame stores, and these frame stores can be of different depth and visual types. In addition, there can be overlay/underlay window pairs in which part of the underlay is visible beneath the overlay. Because the data returned by XGetImage is undefined for portions of the rectangle that have different depths, XGetImage is inadequate to return the picture the user is actually seeing on the screen. In addition,

XGetImage cannot composite pixel information for an overlay/underlay window pair because the pixel information lies in different drawables. XReadScreen addresses these problems.

Rather than returning pixel information, XReadScreen returns color information—the actual displayed colors visible on the screen. The routine returns the color information from any window within the boundaries of the specified rectangle. Unlike XGetImage, the returned contents of visible regions of inferior or overlapping windows of a different depth than the specified window's depth are not undefined. Instead, the actual displayed colors for these windows is returned.

Note – The colors returned are the ones that would be displayed if an unlimited number of hardware color LUTs were available on the screen. Thus, the colors returned are the theoretical display colors. If colormap flashing is present on the screen because there aren't enough hardware color LUTs to display all of the software colormaps simultaneously, the returned colors may be different from the colors that are actually displayed.

The syntax and arguments for this routine are shown below.

```
XImage *
XReadScreen (Display *display, Window w, int x, int y,
        unsigned int width, unsigned int height,
        Bool includeCursor)
```

display	Specifies the connection to the X server.
w	Specifies the window from whose screen the data is read.
x, y	Specify the X and Y coordinates of the upper-left corner of the rectangle relative to the origin of the window w.
width, height	Specify the width and height of the rectangle.
includeCursor	Specifies whether the cursor image is to be included in the colors returned.

If w is an overlay window, the overlay color information is returned wherever there is opaque paint in the specified rectangle. The color information of the underlay is returned wherever there is transparent paint in the overlay. In general, since this underlay can be an overlay window containing transparent paint, the color information for a coordinate (x, y) that contains transparent paint is the youngest non-inferior that has opaque paint at (x, y).

The color data is returned as an XImage structure. The returned image has the same width and height as the arguments specified. The format of the image is ZPixmap. The depth of the image is 24 and the bits_per_pixel is 32. The most significant 8 bits of color information for each color channel (red, green, blue) are returned in the bit positions defined by red_mask, green_mask, and blue_mask in the XImage. The values of the following attributes of the XImage are server dependent: byte_order, bitmap_unit, bitmap_bit_order, bitmap_pad, bytes_per_line, red_mask, green_mask, blue_mask.

If includeCursor is True, the cursor image is included in the returned colors. Otherwise, it is excluded.

Note that the borders of the argument window (and other windows) can be included and read with this request.

If a problem occurs, XReadScreen returns NULL.

Using Existing Xlib Pixel Transfer Routines With Overlay Windows

The Xlib pixel transfer routines XGetImage, XCopyArea, and XCopyPlane can also be used with overlay windows.

XGetImage

On non-overlay drawables, the XGetImage routine works as defined in the X11 specification. The same is true for overlay windows, with the exception that, on these windows, the color information returned for transparent pixels is undefined. Clients who simply want to retrieve the display colors for a region on the screen should use XReadScreen.

XCopyArea and XCopyPlane

When both the source and destination drawables are non-overlay, the `XCopyArea` and `XCopyPlane` routines work as defined in the X11 specification. However, note the following for the cases in which either the source or the destination drawable is an overlay window.

- When the source drawable is overlay and the destination drawable is non-overlay, only the color information is copied; the paint type information in the source is ignored. Color information for transparent pixels is undefined.
- When the source drawable is non-overlay and the destination drawable is overlay, the copy is performed as the paint type in the `GC` indicates. If the paint type is `XSolarisOvlPaintOpaque`, the color information is copied into the destination with opaque paint. If the paint type is `XSolarisOvlPaintTransparent`, the color information is ignored, and the destination pixels are transparent.
- When both the source drawable and destination drawable are overlay, the paint type of the source is ignored, and this behaves as if the source were not an overlay. If copying both color and paint type information is the desired result, use `XSolarisOvlCopyAreaAndPaintType`.

Designing an Application for Portability

The Solaris overlay API provides two routines that help ensure application portability across devices. These routines are:

- `XSolarisOvlSelectPartner` – Enables the application to select the visual that is the best partner for an existing overlay or underlay visual.
- `XSolarisOvlSelectPair` – Enables the application to select the optimal overlay and underlay visual pair from the set of all visual pairs for the screen.

These routines are described below.

Selecting a Visual for an Overlay/Underlay Window

Portable applications using overlays can search for an appropriate overlay visual to use for a given underlay visual, or vice versa. Each X screen supporting the overlay extension defines a set of overlay visuals whose windows are best for use as children of underlay windows. For each underlay visual, there is a set of optimal overlay visuals. Together, all combinations of underlay visuals and their optimal overlay visuals form the set of optimal overlay/underlay pairs for that screen. The overlay and underlay visuals of an optimal pair are partners of each other.

The routine `XSolarisOvlSelectPartner` allows the client to select, given an underlay visual, an optimal overlay that meets certain criteria. Inversely, it also allows the client to select an optimal underlay visual given an overlay visual. The client is assured that, short of X errors not related to overlays, it can successfully create a window with the returned visual.

This routine searches through the optimal partners of the given visual, applying the criteria specified. It returns a success or failure status depending on whether it finds a visual that meets the criteria. A criterion can be one of two types:

1. Hard criterion – A criterion that must be satisfied. Only visuals that meet hard criteria are candidates for successful matches.
2. Soft criterion – A desirable criterion, but one that is not required.

The visual that matches all hard criteria and the most soft criteria is chosen, and its attributes are returned. If two or more visuals are found that meet all of the hard criteria and the same number of soft criteria, one of them will be chosen and returned. It is implementation dependent which one is chosen.

The syntax and arguments for `XSolarisOvlSelectPartner` are shown below.

```
XSolarisOvlSelectStatus
XSolarisOvlSelectPartner (Display *display, int screen,
        VisualID vid, XSolarisOvlSelectType seltype, int numCriteria,
        XSolarisOvlVisualCriteria *pCriteria,
        XVisualInfo *visinfoReturn,
        unsigned long *unmetCriteriaReturn)
```

`display`	Specifies the connection to the X server.
`screen`	An integer specifying the screen for the visual `vid`.
`vid`	The XID of the visual to find a partner for.
`seltype`	The type of selection that is to be done.
`numCriteria`	The number of `XSolarisOvlVisualCriteria` structures in the `pCriteria` array.
`pCriteria`	An array of criteria structures in priority order from high to low specifying the criteria to be used in selecting the visual.
`visinfoReturn`	A pointer to a caller provided `XVisualInfo` structure. On successful return, this structure contains a description of the chosen visual.
`unmetCriteriaReturn`	A pointer to a bitmask that describes the criteria that were not satisfied. This return argument is meaningful only when the routine returns a value of `XSolarisOvlQualifiedSuccess`, or `XSolarisOvlCriteriaFailure`.

Argument Types

`XSolarisOvlSelectType` is an enumeration defining two types of selections that can be done in `XSolarisOvlSelectPartner`. It is defined as:

```
typedef enum {
    XSolarisOvlSelectBestOverlay,
    XSolarisOvlSelectBestUnderlay,
} XSolarisOvlSelectType;
```

`XSolarisOvlVisualCriteria` is a structure defining various criteria to be used during visual selection, along with indications of the stringency of the criteria. This structure is defined as:

```
typedef struct {
    unsigned long      hardCriteriaMask;
    unsigned long      softCriteriaMask
    int                c_class;
    unsigned int       depth;
    unsigned int       minColors;
    unsigned int       minRed;
    unsigned int       minGreen;
    unsigned int       minBlue;
    unsigned int       minBitsPerRGB;
    unsigned int       minBuffers;
} XSolarisOvlVisualCriteria;
```

`hardCriteriaMask` and `softCriteriaMask` are bitmasks whose values can be the logical OR of any of the following bitmasks:

```
#define XSolarisOvlVisualClass       (1L<<0)
#define XSolarisOvlDepth             (1L<<1)
#define XSolarisOvlMinColors         (1L<<2)
#define XSolarisOvlMinRed            (1L<<3)
#define XSolarisOvlMinGreen          (1L<<4)
#define XSolarisOvlMinBlue           (1L<<5)
#define XSolarisOvlMinBitsPerRGB     (1L<<6)
#define XSolarisOvlMinBuffers        (1L<<7)
#define XSolarisOvlUnsharedPixels    (1L<<8)
#define XSolarisOvlUnsharedColors    (1L<<9)
#define XSolarisOvlPreferredPartner  (1L<<10)
```

Return Types

`XSolarisOvlSelectStatus` is a value that indicates whether the routine succeeded in finding a visual and, if it failed, the reason for the failure. The return value can be one of:

```
typedef enum {
    XSolarisOvlSuccess,
    XSolarisOvlQualifiedSuccess,
    XSolarisOvlCriteriaFailure,
    XSolarisOvlFailure,
} XSolarisOvlSelectStatus;
```

- `XSolarisOvlSuccess` is returned if the search is completely successful in finding a visual that meets all hard and soft criteria of one of the `XSolarisOvlVisualCriteria` structure.
- `XSolarisOvlQualifiedSuccess` is returned if the chosen visual satisfies all hard criteria of one of the `XSolarisOvlVisualCriteria` structure, but doesn't meet all soft criteria. In this case, `unmetCriteriaReturn` contains the logical OR of the soft criteria that were not met.
- `XSolarisOvlCriteriaFailure` indicates that no visual could be found that meets all the hard criteria of any of the `XSolarisOvlVisualCriteria` structures. In this case, `unmetCriteriaReturn` contains the logical OR of the hard criteria that were not met for the XSolarisOvlVisualCriteria structure with the fewest hard criteria not met.
- `XSolarisOvlFailure` is returned if some other error is encountered besides criteria match failure.

Multiple Criteria Sets

`XSolarisOvlSelectPartner` supports a *degradation sequence* of criteria sets. This means that multiple criteria sets can be specified in a single call. First, the routine attempts to find a visual matching the first criteria set. If a visual is found that meets all of the hard criteria of the first set, this visual is chosen. If no visual meets all hard criteria of the first set, the routine performs a search using the second criteria set. This process continues until either a visual is found that meets the hard criteria of some criteria set, or all sets have been

used to search. This degradation sequence allows clients to specify the criteria for the most preferred visual as the first criteria set. Visuals that are acceptable but are less desirable can be specified in criteria sets following the first criteria set. This allows the search to proceed through a progressive relaxation in the client's requirements for the visual with a single subroutine call.

Any of the possible criteria can be specified either as a hard or soft criteria for a particular criteria set. For a given set, `hardCriteriaMask` is the logical OR of the criteria bitmasks that are to be applied as hard criteria during the search. Likewise, `softCriteriaMask` is the logical OR of the soft criteria bitmasks.

Some criteria have values associated with them. These values are provided by other data members in the `XSolarisOvlVisualCriteria` structure. In the criteria descriptions that follow, these data members are mentioned where applicable.

- `XSolarisOvlVisualClass` specifies that the client wants the selected visual to have a specific visual class. The required class is specified in `c_class`.
- The following criteria interact within one another: `XSolarisOvlDepth`, `XSolarisOvlMinColors`, `XSolarisOvlMinRed`, `XSolarisOvlMinGreen`, and `XSolarisOvlMinBlue`. Typically only some subset of these should be specified.
- `XSolarisOvlDepth` specifies that the depth of the selected visual is to be equal to `depth`.
- `XSolarisOvlMinColors` specifies that the selected visual is to have at least `minColors` number of total displayable colors.
- `XSolarisOvlMinRed`, `XSolarisOvlMinGreen`, and `XSolarisOvlMinBlue` can be used to indicate more specific color requirements for `DirectColor` or `TrueColor` visuals. Their corresponding values are specified in `minRed`, `minGreen`, and `minBlue`, respectively. These indicate that the selected visual must have at least the specified number of reds, greens, and/or blues.
- `XSolarisOvlMinBitsPerRGB` specifies that the selected visual is to have at least `minBitsPerRGB` of color channel output from colormaps created on that visual.
- `XSolarisOvlMinBuffers` specifies that the client wants the selected visual to be able to be assigned at least `minBuffers` number of accelerated MBX image buffers.

- `XSolarisOvlUnsharedPixels` selects partner visuals whose window pixels don't lie in the same drawing plane groups as the window pixels of the argument visual `vid`. If a visual uses the same drawing plane group as the argument visual, it is not matched by this criterion.
- `XSolarisOvlUnsharedColors` selects partner visuals whose window pixel colors can be displayed simultaneously when the overlay/underlay window pair has the colormap focus. If a visual shares the same color LUT pool and that pool has only one color LUT in it as the argument visual, the visual is not matched by this criterion.

If either `hardCriteriaMask` of a criteria set is to 0, any visual will match that criteria set with a hard match. Likewise, setting the `softCriteriaMask` of a criteria set to `0`, is sufficient to guarantee at least a soft match for that criteria set.

Selecting an Optimal Overlay/Underlay Visual Pair

The `XSolarisOvlSelectPair` routine is similar to `XSolarisOvlSelectPartner`. However, instead of selecting a partner visual given another visual, this routine simultaneously selects both the overlay and underlay visual from the set of all visual pairs for the given screen. The pair selected is the one that best matches the given criteria. The client is assured that, short of X errors not related to overlays, it can successfully create windows with the returned visuals.

This routine searches through all optimal visual pairs for a given screen, and then through all pairs of visuals (optimal and non-optimal), applying the specified criteria. These criteria are specified in `pCriteria`. Each element of `pCriteria` specifies criteria for both the overlay and underlay. It returns a success or failure status depending on whether it finds a pair that meets all the given criteria.

The selected pair has an overlay that satisfies all the hard criteria specified for the overlay. The pair has an underlay visual that satisfies all the hard criteria for the underlay. The attributes of the overlay visual are returned in `ovVisinfoReturn`. Likewise, the attributes of the underlay visual are specified in `unVisinfoReturn`. If two or more pairs are found that meet all of the hard criteria (both overlay and underlay) and the same number of soft criteria (either overlay or underlay), one of them will be chosen and returned. Which pair is chosen depends on the implementation.

The syntax and arguments are shown below.

```
XSolarisOvlSelectStatus
XSolarisOvlSelectPair (Display *display, int screen, int
numCriteria,
       XSolarisOvlPairCriteria *pCriteria,
       XVisualInfo *ovVisinfoReturn, XVisualInfo *unVisinfoReturn,
       unsigned long *unmetOvCriteriaReturn,
       unsigned long *unmetUnCriteriaReturn)
```

`display`	Specifies the connection to the X server.
`screen`	An integer specifying the screen on which the visuals are to be searched.
`numCriteria`	The number of `XSolarisOvlPairCriteria` structures in the `pCriteria` array.
`pCriteria`	An array of pair criteria structures in priority order from high to low specifying the criteria to be used in selecting the pair.
`ovVisinfoReturn`	A pointer to a caller-provided `XVisualInfo` structure. On successful return, this structure contains a description of the chosen overlay visual.
`unVisinfoReturn`	A pointer to a caller-provided `XVisualInfo` structure. On successful return, this structure contains a description of the chosen underlay visual.
`unmetOvCriteriaReturn`	A pointer to a bitmask that describes the criteria that were not satisfied for the overlay visual. This return argument is meaningful only when the routine returns a value of `XSolarisOvlQualifiedSuccess`, or `XSolarisOvlCriteriaFailure`.

`unmetUnCriteriaReturn` A pointer to a bitmask that describes the criteria that were not satisfied for the underlay visual. This return argument is meaningful only when the routine returns a value of `XSolarisOvlQualifiedSuccess`, or `XSolarisOvlCriteriaFailure`.

Argument Types

`XSolarisOvlPairCriteria` is a structure defining various criteria to be used during visual selection, along with indications of the stringency of the criteria. This structure is defined as:

```
typedef struct {
    XSolarisOvlVisualCriteriaoverlayCriteria;
    XSolarisOvlVisualCriteriaunderlayCriteria;
} XSolarisOvlPairCriteria;
```

`XSolarisOvlVisualCriteria` is defined in the specification of `XSolarisOvlSelectPartner`.

Return Types

Refer to the specification of `XSolarisOvlSelectPartner` for the definition of the type `XSolarisOvlSelectStatus`.

- `XSolarisOvlSuccess` is returned if the search is completely successful in finding a pair that meets all hard and soft criteria of one of the `XSolarisOvlPairCriteria` structures.
- `XSolarisOvlQualifiedSuccess` is returned if the chosen pair satisfies all hard criteria of one of the `XSolarisOvlPairCriteria` structures, but doesn't meet all soft criteria. In this case, `unmetOvCriteriaReturn` and `unmetUnCriteriaReturn` contain the logical OR of the soft criteria that were not met for the overlay and underlay, respectively.
- `XSolarisOvlCriteriaFailure` indicates that no pair could be found that meets all the hard criteria of any of the `XSolarisOvlPairCriteria` structures. In this case, `unmetOvCriteriaReturn` and

`unmetUnCriteriaReturn` contain the logical OR of the hard criteria that were not met by the `XSolarisOvlPairCriteria` structure with the fewest hard failures, for the overlay and underlay, respectively.

- `XSolarisOvlFailure` is returned if some other error is encountered besides criteria match failure.

Criteria Sets

Like `XSolarisOvlSelectPartner`, `XSolarisOvlSelectPair` supports a *degradation sequence* of criteria sets. This means that multiple criteria sets can be specified in a single call. First, the routine attempts to find a pair matching the first criteria set for both the overlay and the underlay. If it finds a pair that meets all of the hard criteria of the first set, it chooses this pair. If no pair meets all hard criteria of the first set, the routine searchs using the second criteria set. This process continues until either a pair is found that meets all of the hard criteria of some criteria set, or all sets have been used to search. This degradation sequence allows clients to specify the criteria for the most preferred pair as the first criteria set. Pairs that are acceptable but less desirable can be specified in criteria sets following the first criteria set. This allows the search to proceed through a progressive relaxation in the client's requirements for the pair with a single subroutine call.

The criteria masks that can be specified are described in "Selecting a Visual for an Overlay/Underlay Window" on page 83.