The Implementation of Graphics Facilities in Icon Version 9 *

Clinton L. Jeffery

Technical Report CS-94-3

August 25, 1994

Abstract

The graphics facilities in Icon Version 9 are a large addition to the Icon system. This document describes the internals of the graphics implementation. It is intended primarily for those porting the graphics and windowing facilities to a new window system.

Division of Mathematics, Computer Science, and Statistics
The University of Texas at San Antonio
San Antonio, TX 78249

*This work was supported in part by the National Science Foundation under Grants CCR-8713690 and CCR-8901573, a grant from the AT&T Research Foundation, a software donation from Microsoft, and a UTSA Faculty Research Award.

Introduction

This document describes the internals of the implementation of Icon's graphics and window system facilities. Much of the code is devoted to hiding specific features of C graphics interfaces that were deemed overly complex or not worth the coding effort they entail. Other implementation techniques are motivated by portability concerns. The graphics interface described below has been implemented to various levels of completeness under X Window, OS/2 Presentation Manager, Microsoft Windows, and Macintosh platforms.

The X Implementation

The reference implementation of Icon's graphics facilities is written in terms of Xlib, the lower-level X Window C interface [Nye88]. It does not use the X resource manager. The end result of these two facts is that the implementation is relatively visible: the semantics are expressed fairly directly in the source code. Although it is necessary to understand the semantics of the underlying X routines, hidden behavior has been minimized.

The X implementation employs the XPM X pixmap library if it is available; XPM is a proposed extension to Xlib for storing color images in external files [LeHo91]. XPM provides color facilities analogous to the built-in X black-and-white bitmap routines. In addition to the image formats native to each platform, Icon also supports GIF as a more universal image file format.

Relevant Source File Summary

This document assumes a familiarity with the general organization and layout of Icon sources and the configuration and installation process. For more information on these topics, consult Icon Project Documents IPD 238 [Gris94a] and IPD 243 [Gris94b]. Icon's graphics facilities consist of several source files, all in the runtime directory unless otherwise noted. They are discussed in more detail later in this document.

header files – h/graphics.h contains structures and macros common across platforms. Each platform adds platform-specific elements to the common window structures defined in this file. In addition, each platform gets its own header file, currently these consist of X Windows (h/xwin.h), Presentation Manager (h/pmwin.h), Microsoft Windows and Windows NT (h/mswin.h) and the Macintosh (h/mac.h). Every platform defines several common macros in the window-system specific header file in addition to its window system specific structures and macros. The common macros are used to insert platform-dependent pieces into platform-independent code.

Icon functions – fwindow.r contains the RTL (Run-Time Language) interface code used to define built-in functions and operators for the Icon interpreter and compiler. This file is entirely window system independent.

internal support routines – rwindow.r, rwinrsc.r, rgfxsys.r, and rwinsys.r are basically C files with some window system dependencies but mostly consisting of code that is used on all systems.

window-system specific files – Each window system gets its own source files for C code, included by the various r*.r files in the previous section. Currently these include rxwin.ri and rxrsc.ri for X Window; rpmwin.ri, rpmrsc.ri, and rpmgraph.ri for Presentation Manager; rmswin.ri for MS Windows and Windows NT; and rmac.ri for the Macintosh. Each platform will implement one or more such r*.ri files. In addition, common/xwindow.c contains so many X Window includes that it won't even compile under UNIX Sys V/386 R 3.2 if all of the Icon includes are also present – so its a .c file instead of a .r file.

Tainted "regular" Icon sources – Many of the regular Icon source files include code under #ifdef Graphics and/or one or more specific window system definitions such as #ifdef XWindows or #ifdef PresentationManager. The tained files that typically have to be edited for a new window system include h/grttin.h, h/features.h, h/rexterns.h, h/rmacros.h, h/rproto.h, h/rstructs.h, and h/sys.h. Other files also contain Graphics code. This means that most of the system has to be recompiled with rtt and cc after Graphics is defined in h/define.h. You will also want to study the Graphics stuff at the end of h/grttin.h since several profound macros are there. Also, any new types (such as structures) defined in your window system include files will need dummy declarations (of the form typedef int foo;) to be added there.

Under UNIX the window facilities are turned on at configuration time by typing make X-Configure name=system instead of the standard make Configure invocation. The X configuration modifies makefiles and defines the symbolic constant Graphics in h/define.h. Analogous configuration handling is performed for other systems; for example, an alternate .bat file is used in place of os2.bat or turbo.bat.

Graphics #define-d symbols

The primary, window-system-independent defined symbol that turns on window facilities is simply Graphics. Underneath this parent #ifdef, the symbol XWindows is meant to mark all X Window code. Other window systems have a definition comparable to XWindows: for Microsoft Windows, MSWindows, for OS/2, PresentationManager, and for the Macintosh, MacGraph. Turning on any of the platform specific graphics #define symbols turns on Graphics implicitly.

Structures Defined in graphics.h

graphics.h defines a collection of C structures that contain pointers to other C structures from graphics.h as well as pointers into the window system library structures. The internals for the simplest Icon window structure are depicted in Figure 1.

At the top, Icon level, there is a simple structure called a binding that contains a pointer to a window state and a window context. Pointers to bindings are stored in the FILE * variable of the Icon file structure, and most routines that deal with a window take a pointer to a binding as their first argument. Beneath this facade, several structures are accessed to perform operations on each window.

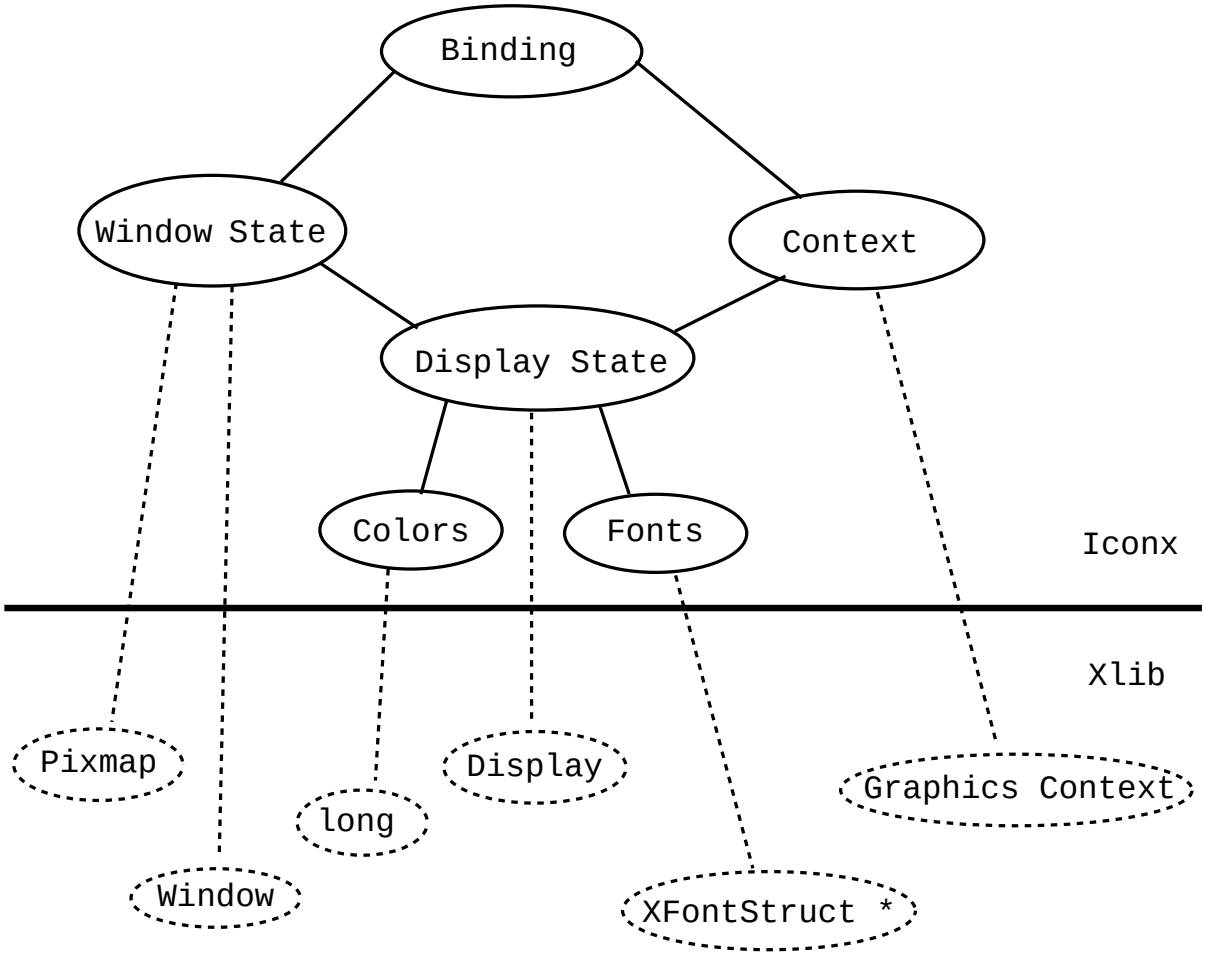

Figure 1: Internal Structure of an Icon Window Value

The window state holds the typical window information (size, text cursor location, an Icon list of events waiting to be read) as well as a window system pointer to the actual window, a pointer to a backing pixmap (a "compatible device context" used to handle redraw requests), and a pointer to the display state.

The window context contains the current font, foreground, and background colors used in producing output on the canvas. It also contains drawing style attributes such as the *fill style*. Contexts are separate from the window state so that they may be shared among windows. This is a big win, and Icon programs tend to use it heavily, so in porting the window functions a central design issue must be the effective use of a comparable facility on other window systems, or emulating the context abstraction if necessary. Nevertheless, one might start out with Couple() and Clone() disabled and only allow one hardwired context associated with each window.

The display state contains whatever system resources (typically pointers or handles) that are shared among all the windows on a given display in the running program. For example, in X this includes the fonts,

the colors, and a window system pointer for an internal Display structure required by all X library calls to denote the connection to the X server.

Macros and Coding Conventions Defined in the Window System Header

Since the above structure is many layers deep and sometimes confusing, Icon's graphics interface routines employ coding conventions to simplify things. In order to avoid many extra memory references in accessing fields in the multi-level structure, "standard" local variables are declared in most of the interface routines in rwindow.r. The macro STDLOCALS(w) declares local variables pointing to the most commonly used pieces of the window binding, and initializes them from the supplied argument; each window system header should define an appropriate STDLOCALS(w) macro.

Some common standard locals (taken from the X Window implementation) are wc, ws, wd, stdgc, stddpy, stdwin, and stdpix. While wc, ws and wd are pointers to structures copied from the window binding, stdgc, stddpy, stdwin, and stdpix are actual X entities that are frequently supplied to the Xlib routines as arguments. Other window systems may have more or fewer standard locals.

In much of the source code, the window system calls are performed twice. This is because many platforms such as X and PresentationManager do not remember the contents of windows when they are reduced to iconic size or obscured behind other windows. When the window is once again exposed, it is sent a message to redraw itself. Icon hides this entirely, and remembers the contents of the window explicitly in a window-sized bitmap of memory. The calling of platform graphics routines twice is so common that a set of macros is defined in xwin.h to facilitate it. The macros are named RENDER2 through RENDER6, and each of them takes an Xlib function and then some number of arguments to pass that function, and then calls that function twice, once on the window and once on the bitmap.

Platforms that provide backing store may avoid this duplicated effort. In practice however it seems most window systems have redraw events even if they implement retained windows (for example, MGR [Uhle88]).

Window Manipulation in rxwin.ri

Most of the actual Xlib calls are in rxwin.ri in the Icon run-time system directory. This module includes routines in several major areas:

- Window creation and destruction
- Low-level event processing
- Low-level text output operations
- Window and context attribute manipulation

Window Creation and Destruction

A graphics window is created when the Icon programmer calls open() with file attribute "g". The window opening sequence consists of a call to wopen() to allocate appropriate Icon structures for the window and evaluate any initial window attributes given in additional arguments to open(). After these attributes have been evaluated, platform resources such as fonts and colors are allocated and and the window itself is instantiated. wopen() busy-waits until the window has received its first expose event, ensuring that no subsequent window operation takes place before the window has appeared onscreen.

A window is closed by a call to wclose(); this removes the on-screen window even if other bindings (Icon window values) refer to it. A closed window remains in memory until all Icon values that refer to it are closed. A call to unbind() removes a binding without necessarily closing the window.

Event Processing

The system software for each graphics platform has a huge number of different types of events. Icon ignores most of them. Of the remainder, some are handled by the runtime system code in the .ri files implicitly, and some are explicitly passed on to the Icon program.

Most native graphic systems require that applications be event-driven; they must be tightly I/O bound around the user's actions. The interaction between user and program must be handled at every instant by the program. Icon, on the other hand, considers this event-driven model optional.

Making the event-driven model optional means that the Icon interface must occasionally read and process events when the Icon program itself is off in some other computation. In particular, keystrokes and mouse events must be stored until the user requests them, but exposure events and resizes must be processed immediately. The Icon interpreter pauses at regular intervals in between its virtual machine instructions[1] and polls the system for events that must be processed; this technique fails when no virtual machine instructions are executing, such as during garbage collections or when blocked on file I/O.

This probably could be done using the Xlib event queue manipulation routines. Instead, the Icon window interface maintains its own keystroke and mouse event queue from which the Icon functions obtain their events. This additional queue makes the implementation more portable (various window systems probably do not support queue manipulation to the extent or in the same way that X does). It also provides the programmer with a higher level event processing abstraction which has proven useful.

Window resizing is partly handled by the interface. The old contents of the window are retained in their original positions, but the program is informed of the resize so it can handle the resize in a more reasonable manner. As has already been noted exposure events are hidden entirely via the use of a backing pixmap with identical contents for each window. The pixmap starts out the same size as the window. It is resized whenever the window grows beyond one of its dimensions. It could be reduced whenever the window shrinks, but then part of the window contents would be lost whenever the user accidentally made the window smaller than intended.

rwindow.r also contains tables and routines mapping strings for various attributes and values to native window system integer constants. Binary search is employed. This approach is far from satisfactory, but in

[1] the Icon compiler emits polling code in its generated C code, so window system facilities are supported by the compiler as well

the absence of language support for integer constants it is an adequate way to provide symbolic access "built-in" to the language. Additional tables mapping strings to integers are found in the window-system-specific source files.

Memory Management and r*rsc.ri Files

Memory management for internal window structures is independent of Icon's standard memory management system. Xlib memory is allocated using *malloc(2)*. Most internal Icon window structures could be allocated in Icon's block region, but since they are acyclic and cannot contain any pointers to Icon values, this would serve little purpose [2]. In addition when an incoming event is being processed it has to be matched up with the appropriate window state structure, so some of the window structures must be easily reached, not lost in the block region. The window interface structures are reference counted and freed when the reference count reaches 0.

Color Management

Managing colors under X Windows is painful. In particular, if the same color is allocated twice the color table entry is shared (which is good) and that entry may only be freed once (which is bad). For this reason, every color allocated by Icon is remembered and duplicate requests are identified and freed only once. In the general case it is impossible to detect when a particular color is no longer being displayed, and so colors are only freed on window closure or when a window is cleared.

Font Management

Icon supports a portable font name syntax. Since the available fonts on systems vary widely, "interesting" code has been written to support these portable names on various X servers. Each window system may need to include heuristics to pick an appropriate font in the font allocation routine in the window system's r*.ri file.

External Image Files

Reading and writing window contents to external files is accomplished by the routines loadimage() and dumpimage(), implemented in each platform's window system specific file, such as rxwin.ri. These routines take a window binding and a string filename and perform the I/O transfer. Presently, the file format is assumed to be indicated by the filename extension; this is likely to change. Ideally Icon should tolerate different file formats more flexibly, inferring input file formats by reading the file header where possible, and running external conversion programs where appropriate. GIF files are self-identifying, so they are always recognized independent of name.

[2] Actually, it is probably the right thing to do, and will probably happen some day, but its just not in the cards right now unless *you* feel like messing with the garbage collector.

Porting Reference

This section documents the window-system specific functions and macros that generally must be implemented in order to port Icon's graphics facilities to a new window system. The list is compiled primarily by studying fwindow.r, rwindow.r, and the existing platforms.

A note on types: w is a window binding pointer (wbp), the top level Icon "window" value. i is an integer, s is a string. wsp is the window state (a.k.a. canvas) pointer, and wcp is the window context pointer. A bool return value returns one of the C macro values Succeeded or Failed.

ANGLE(a)

Convert from radians into window system units. For example, under X these are 1/64 of a degree integer values, while under PresentationManager it converts to units of 1/65536 of a degree in a fixed-point format.

ARCHEIGHT(arc)

The height component of an XArc

ARCWIDTH(arc)

The width component of an XArc

ASCENT(w)

Returns the number of pixels above the baseline for the current font. Note that when Icon writes text, the (x,y) coordinate gives the left edge of the character at its baseline; some window systems may need to translate our coordinates.

---

int blimage(w, x, y, width, height, ch, s, len)

Draws a bi-level (i.e. monochrome, 1-bit-per-pixel) image; used in DrawImage() which draws bitmap data stored in Icon strings.

---

wcp clone_context(w)

Allocate a new context, cloning attributes from w's context.

COLTOX(w, i)

Return integer conversion from a 1-based text column to a pixel coordinate

---

copyArea(w1,w2,x,y,width,height,x2,y2)

Copies a rectangular block of pixels from w1 to w2.

DESCENT(w)

Returns the number of pixels below the baseline for the current font.

DISPLAYHEIGHT(w)

Return w's display height in pixels.

DISPLAYWIDTH(w)

Return w's display width in pixels.

---

bool do_config(w, i)

Performs move/resize operations after one or more attributes have been evaluated. Config is a word with two flags: the one bit indicates a move, the two bit indicates a resize. The desired sizes are in the window state pointer, e.g. w-¿window-¿width.

---

drawarcs(w, thearcs, i)

Draw i arcs on w, given in an array of XArc structures. Define an appropriate XArc structure for your window system; it must include fields x, y and width and height fields accessible through macros ARCWIDTH() and ARCHEIGHT(). Also, a starting angle angle1 and arc extent angle2, assigned through macros ANGLE(), EXTENT(), and FULLARC. This is currently a mess. Imitation of the X or PresentationManager code is in order.

drawlines(w, points, i)

Draw i−1 connected lines, connecting the dots given in points.

---

drawpoints(w, points, i)

Draw i points.

---

drawsegments(w, segs, i)

Draw i disconnected line segments; define an XSegment structure appropriate do your window system, consisting of fields x1, y1, x2, y2. This type definition requirement should be cleaned up someday.

---

drawstring(w, x, y, s, s_len)

Draw string s at coordinate (x,y) on w. Note that y designates a baseline, not an upper-left corner, of the string.

---

drawrectangles(w, rectangles, i)

Draw i rectangles. Define an XRectangle structure appropriate to your window system.

---

int dumpimage(w, s, x, y, width, height)

Write an image of a rectangular area in w to file s. Returns 0 for failure; should change to Succeeded/Failed.

---

eraseArea(w, x, y, width, height)

Erase a rectangular area, that is, set it to the current background color. Compare with and fillrectangles().

EXTENT(a)

Convert from radians into window system units, e.g. under PresentationManager it converts to units of 1/65536 of a circle and does some weird type conversion.

---

fillarcs(w, arcs, i)

Fill wedge-like arc sections (pie pieces). See drawarcs().

---

fillrectangles(w, rectangles, i)

Fill i rectangles. See drawrectangles().

---

fillpolygon(w, points, i)

Fill a polygon defined by i points. Connect first and last points if they are not the same.

FHEIGHT(w)

Returns the pixel height of the current font, hopefully ASCENT + DESCENT.

---

free_binding(w)

Free binding associated with w. This gets rid of a binding that refers to w, without necessarily closing the window itself (other bindings may point to that window).

---

free_context(wc)

Free window context wc.

---

free_mutable(w, i)

Free mutable color index i.

---

free_window(ws)

Free window canvas ws.

---

freecolor(w, s)

Free a color allocated on w's display.

FS_SOLID

Define this to be the window system's solid fill style symbol.

FS_STIPPLE

Define this to be the window system's stippled fill style symbol.

FULLARC

Window-system value for a complete (360 degree) circle or arc.

FWIDTH(w)

Returns the pixel width of the widest character in the current font.

---

wsp getactivewindow()

Return a window state pointer to an active window, blocking until a window is active. Probably will be generalized to include a non-blocking variant. Returns NULL if no windows are opened.

---

getbg(w, s)

Returns (writes into s) the current background color.

getcanvas(w, s)

Returns (writes into s) the current canvas state.

---

getdefault(w, s_prog, s_opt, s)

Get any window system defaults for a program named s_prog resource named s_opt, write result in s.

---

getdisplay(w, s)

Write a string to s with the current display name.

---

getdrawop(w, s)

Return current drawing operation, one of various logical combinations of source and destination bits.

---

getfg(w, s)

Returns (writes into s) the current foreground color.

---

getfntnam(w, s)

Returns (writes into s) the current font. This interface may get changed since a portable font naming mechanism is to be installed. Name is presently always prefixed by "font=" (pretty stupid, huh); must be an artifact of merging window system ports, will be changed.

---

geticonic(w, s)

Return current window iconic state in s, could "iconify" or whatever. Obsolete (subsumed by canvas attribute, getcanvas()).

---

geticonpos(w, s)

Return icon's position to s, an encoded "x,y" type string.

---

int getimstr(w, x, y, width, height, paltbl, data)

Gets an image as a string. Used in GIF code.

---

getlinestyle(w, s)

Return current line style, one of solid, dashed, or striped.

---

get_mutable_name(w, i)

Returns the string color name currently associated with a mutable color.

---

getpattern(w, s)

Return current fill pattern in s.

---

getpixel(w, x, y, long *rv)

Assign RGB value for pixel (x,y) into *rv.

---

getpixel_init(w, x, y, width, height)

Prepare to fetch pixel values from window, obtaining contents from server if necessary.

---

getpointername(w, s)

Write mouse pointer appearance, by name, to s.

---

getpos(w)

Update the window state's posx and posy fields with the current window position.

---

getvisual(w, s)

Write a string to s that explains what type of display w is on, e.g. "visual=x,y,z", where x is a class, y is the bits per pixel, and z is number of colormap entries available. This terrible X-specific anachronism is going to go away.

HideCursor(wsp ws)

Hide the text cursor on window state ws.

ICONFILENAME(w)

Produce char * for window's icon image file name if there is one.

ICONLABEL(w)

Produce char * for icon's title if there is one.

---

isetbg(w, i)

Set background color to mutable color table entry i. Mutable colors are not available on all display types.

---

isetfg(w, i)

Set foreground color to mutable color table entry i. Mutable colors are not available on all display types.

ISICONIC(w)

Return 1 if the window is presently minimized/iconic, 0 otherwise.

ISFULLSCREEN(w)

Return 1 if the window is presently maximized/fullscreen, 0 otherwise.

ISNORMALWINDOW(w)

Return 1 if the window is neither minimized nor maximized, 0 otherwise.

LEADING(w)

Return current integer leading, the number of pixels from line to line.

LINEWIDTH(w)

Return current integer line width used during drawing.

---

lowerWindow(w)

Lower the window to the bottom of the stack.

---

mutable_color(w, dptr dp, i, C_integer *result)

Allocate a mutable color from color spec given by dp and i, placing result (a small negative integer) in *result.

---

nativecolor(w, s, r, g, b)

Interpret a platform-specific color name s (define appropriately for your window system). Under X, we can do this only if there is a window.

---

pollevent()

Poll for available events on all opened displays. This is where the interpreter calls the window system interface. Return a -1 on an error, otherwise return count of how long before it should be polled (400).

---

query_pointer(w, XPoint *xp)

Produce mouse pointer location relative to w.

---

query_rootpointer(XPoint *xp)

Produce mouse pointer location relative to root window on default screen.

---

raiseWindow(w)

Raise the window to the top of the stack.

---

bool readimage(w, s, x, y, int *status)

Read image from file s into w and (x,y). Status is 0 if everything was kosher, 1 if some colors weren't available but the image was read OK; if a major problem occurs it returns Failed.

---

rebind(w, w2)

Assign w's context to that of w2.

RECHEIGHT(rec)

The height component of an XRectangle. Gets "fixed up" (converted) into an Y2 value if necessary, in window system specific code.

RECWIDTH(rec)

The width component of an XRectangle. Gets “fixed up” (converted) into an X2 value if necessary, in window system specific code.

RECX(rec)

The x component of an XRectangle

RECY(rec)

The y component of an XRectangle

ROWTOY(w, i)

Return integer conversion from a 1-based text row to a pixel coordinate

---

SCREENDEPTH(w)

Returns the number of bits per pixel.

---

int setbg(w, s)

Set the context background color to s. Returns 0 for failure, 1 for success. Should be changed to Succeeded/Failed.

---

setcanvas(w, s)

Set canvas state to s, make it “iconic”, “hidden” or whatever.

---

setclip(w)

Set (enable) clipping on w from its context.

---

setcursor(w, i)

Turn text cursor on or off. Text cursor is off (invisible) by default.

---

setdisplay(w, s)

Set the display to use for this window; fails if the window is already open somewhere.

---

setdrawop(w, s)

Set drawing operation to one of various logical combinations of source and destination bits.

---

int setfg(w, s)

Set the context foreground color to s. Returns 0 for failure, 1 for success. Should be changed to Succeeded/Failed.

---

setfillstyle(w, s)

Set fill style to solid, masked, or textured.

---

bool setfont(w, char **s)

Set the context font to s. This function first attempts to use the portable font naming mechanism; it resorts to the system font mechanism if the name is not in portable syntax.

---

setgamma(w, gamma)

Set the context's gamma correction factor

---

setgeometry(w, s)

Set the window's size and/or position

---

setheight(w, i)

Set window height to i, whether or not window is open yet.

---

seticonicstate(w, s)

Set window iconic state to s, could "iconify" or whatever. Obsolete; setcanvas() is more important.

---

seticonimage(w, dptr d)

Set window icon to d. Could be string filename or existing pixmap (i.e. another window's contents). Pixmap assignment no longer possible, so one could simplify this to just take a string parameter.

---

seticonlabel(w, s)

Set icon's string title to s.

---

seticonpos(w, s)

Move icon's position to s, an encoded "x,y" type string.

---

setimage(w, s)

Set an initial image for the window from file s. Only valid during call to open().

---

setleading(w, i)

Set line spacing to i pixels from line to line. This includes font height and external leading, so $i <$ fontheight means lines draw partly over preceding lines, $i >$ fontheight means extra spacing.

---

setlinestyle(w, s)

Set line style to solid, dashed, or striped.

---

setlinewidth(w, i)

Set line width to i.

---

set_mutable(w, i, s)

Set mutable color index i to color s.

SetPattern(w, s, s_len)

Set fill pattern to bits given in s. Fill pattern is not used unless fillstyle attribute is changed to “patterned” or “opaquepatterned”.

SetPatternBits(w, width, bits, nbits)

Set fill pattern to bits given in the array of integers named bits. Fill pattern is not used unless fillstyle attribute is changed to “patterned” or “opaquepatterned”.

---

setpointer(w, s)

Set mouse pointer appearance to shape named s.

---

setpos(w, s)

Move window to s, a string encoded “(x,y)” thing.

---

setwidth(w, i)

Set window width to i, whether or not window is open yet.

---

setwindowlabel(w, s)

Set window’s string title to s.

ShowCursor(wsp ws)

Show the text cursor on window state ws.

---

int strimage(w, x, y, width, height, e, s, len)

Draws a character-per-pixel image, used in DrawImage(). See blimage().

SysColor

Define this type to be the window system's RGB color structure.

TEXTWIDTH(w, s, s_len)

Returns the integer text width of s using w's current font.

---

toggle_fgbg(w)

Swap the foreground and background on w.

---

unsetclip(w)

Disable clipping on w from its context.

UpdateCursorPos(wsp ws, wcp wc)

Move the text cursor on window state ws and context wc.

---

walert(w, i)

Sounds an alert (beep). i is a volume; it can range between -100 and 100; 0 is normal.

---

warpPointer(w, x, y)

Warp the mouse location to (x,y)

---

wclose(w)

Closes window w. If there are other bindings that refer to the window, they are converted into pixmaps, i.e. the window disappears but the canvas is still there and can be written on and copied from.

---

wflush(w)

Flush output to window w; a no-op on some systems.

---

wgetq(w, dptr result)

Get an event from w's pending queue, put results in descriptor *res. Returns -1 for an error, 1 for success (should fix this).

WINDOWLABEL(w)

Produce char * for window's title if there is one.

FILE * wopen(s, struct b_list *lp, dptr attrs, i, int *err_index)

Open window named s, with various attributes. This ought to be merged from various window system dependent files, but presently each one defines its own. Copy and modify from rxwin.ri or rpmwin.ri. The return value is really a wbp, cast to a FILE *.

---

wputc(c, w)

Draw character c on window w, interpret newlines, carriage returns, tabs, deletes, backspaces, and the bell.

---

wsync(w, i)

Synchronize server and client (a no-op on some systems). i is a 1 if pending events are to be dropped, a 0 if not. Should be cleaned up.

---

xdis(w, s, s_len)

Draw string s on window w, low-level.

XTOCOL(w, i)

Return integer conversion from a 0-based pixel coordinate to text column.

YTOROW(w, i)

Return integer conversion from a 0-based pixel coordinate to text row.

References

[Gris94a] Griswold, R. E., Jeffery, C. L., and Townsend, G. M. Configuring the Source Code for Version 9.0 of Icon. Technical Report IPD238, Department of Computer Science, University of Arizona, May 1994.

[Gris94b] Griswold, R. E., Jeffery, C. L., and Townsend, G. M. Installing Version 9.0 of Icon on UNIX Platforms. Technical Report IPD243, Department of Computer Science, University of Arizona, June 1994.

[LeHo91] LeHors, A. *The X PixMap Format*. Groupe Bull, Koala Project, INRIA, France, 1991.

[Nye88] Nye, A., editor. *Xlib Reference Manual*. O'Reilly & Associates, Inc., Sebastopol, California, 1988.

[Uhle88] Uhler, S. A. MGR — C Language Application Interface. Technical report, Bell Communications Research, July 1988.