Graphics Facilities for the Icon Programming Language

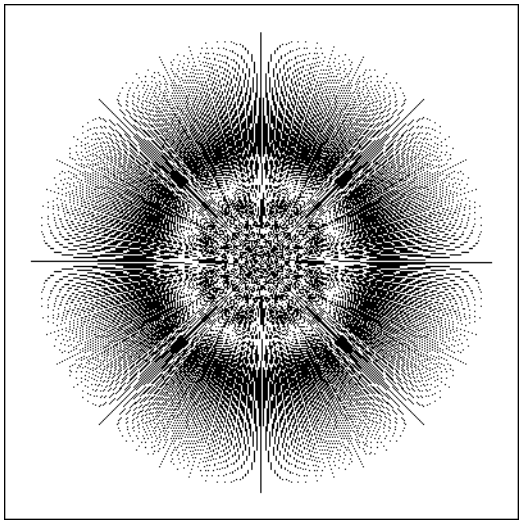
Version 9.3

Gregg M. Townsend
Ralph E. Griswold

Department of Computer Science
The University of Arizona

Clinton L. Jeffery

Division of Computer Science
The University of Texas at San Antonio



IPD281

November 25, 1996

Introduction

The Icon programming language [1] provides a large set of platform-independent facilities for graphical input and output. The implementation includes numerous functions and keywords specifically for graphics. These are augmented by additional library procedures that add higher-level capabilities.

This document describes the graphics facilities of Version 9.1 of Icon [2]; differences from earlier versions are noted in Appendices G and H. A knowledge of Icon is assumed. Previous experience with computer graphics is helpful.

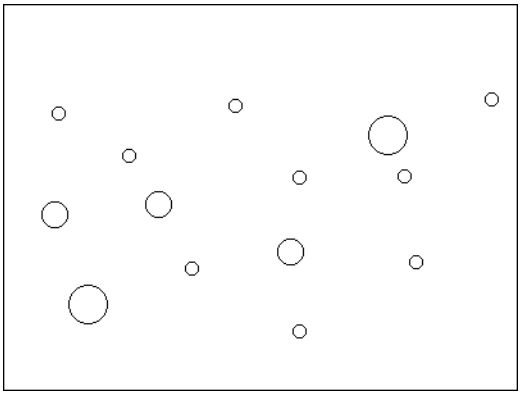
The body of the text presents a survey Icon's graphics capabilities. Full descriptions of the functions, attributes, and other items appear in appendices. The Visual Interface Builder, VIB, is described in a separate document [3].

A Simple Example

This small program illustrates several aspects of graphics programming in Icon:

```
link graphics

procedure main()
   WOpen("size=400,300") | stop("can't open window")
   repeat case Event() of {
      "q":              exit()
      &lpress:          DrawCircle(&x, &y, 5)
      &mpress:          DrawCircle(&x, &y, 10)
      &rpress:          DrawCircle(&x, &y, 15)
      }
end
```



The image at the right shows what the window might look like after several clicks of the mouse.

The link graphics directive gives access to the standard set of graphics procedures in the Icon Program Library [4] that supplement the built-in repertoire.

The WOpen() call creates a window. The window is 400 pixels wide and 300 pixels high, with all other characteristics set by default.

The main loop repeatedly calls Event() to receive input. An event code of "q" is returned when the q key is pressed; when this happens, the program exits.

An event code matching &lpress, &mpress, or &rpress is returned when the left, middle, or right mouse button is pressed. In response, the program draws a circle at the mouse location, obtaining this location from the keywords &x and &y. Circles of radius 5, 10, or 15 pixels are drawn depending on which mouse button was pressed.

Fundamentals

A window in Icon is an object of type window. The value of the keyword &window is the *subject window*. By default, almost all of the graphics functions use the subject window, and a typical program makes no explicit mention of any window value.

When a window is given explicitly as the first argument to a graphics procedure, the procedure and the remaining arguments apply to that window. If the first argument of a graphics procedure is not a window, the procedure and its arguments apply to the subject window. A null first argument to a graphics procedure does *not* specify the subject window; it is illegal.

Graphics actions are accomplished by calling built-in functions and library procedures. The distinction usually is unimportant and is indicated only in Appendix A.

Many drawing functions accept extra sets of parameters to allow multiple drawing operations in a single call. These functions are indicated by the notation "....." in Appendix A.

Most output is drawn using the *foreground color*, which is set by calling Fg(). A few operations make use of the *background color* set by Bg().

Attributes

Window attributes describe and control various characteristics of a window. A full list of these attributes appears in Appendix C.

WAttrib() reads and writes attributes. For example, WAttrib("fg") returns the value of the fg attribute, which is the current foreground color. WAttrib("fg=brown", "linewidth=3") assigns values to the fg and linewidth attributes. WAttrib() ignores invalid names and values.

Some attributes can also be read or written using procedures such as Fg(), Bg(), and Font().

Coordinate System

Window locations and distances are measured in pixels. Angles are measured in radians.

The origin (0,0) is the pixel in the upper-left corner of the window. A left-handed coordinate system is used: x increases towards the right and y increases toward the bottom. As a consequence, angles are measured in a clockwise direction.

Rectangular areas are specified by four integers (x,y,w,h) giving the coordinates of a starting corner and the rectangle's width and height. w and/or h can be negative to extend the rectangle leftward or upward from (x,y).

The effective origin can be moved from (0,0) to (dx,dy) by setting the dx and dy attributes.

Output to the screen is limited to a clipping region that is set by calling Clip(x, y, w, h). Drawing outside the clipping region is not an error, but no pixels outside the region are changed. Clipping is disabled by calling Clip() with no arguments; this is the initial state of a new window.

Drawing Operations

DrawPoint(x, y) draws a single point.

DrawLine(x1, y1, x2, y2, ..., xn, yn) draws a line from (x1,y1) to (x2,y2). If more coordinates are given, (x2,y2) is then connected to (x3,y3), and so on.

DrawPolygon(x1, y1, x2, y2, ..., xn, yn) functions like DrawLine() with the addition that (xn,yn) is connected to (x1,y1) to form a closed path.

DrawSegment(x1, y1, x2, y2) draws a line from (x1,y1) to (x2,y2). Additional pairs of coordinates can be provided to draw additional, disconnected segments.

DrawCurve(x1, y1, x2, y2, ..., xn, yn) draws a smooth curve that passes though the given points. If the first and last points are identical, a smoothly closed curve is produced.

DrawRectangle(x, y, w, h) draws a rectangle having corners at (x,y) and (x+w,y+h).

DrawCircle(x, y, r, theta, alpha) draws a circle or circular arc of radius r centered at (x,y). theta specifies the starting angle, in radians, and alpha is the angle (positive or negative) subtended by the arc. If alpha is omitted, theta is immaterial and a full circle is drawn.

DrawArc(x, y, w, h, theta, alpha) draws an elliptical arc inscribed in the rectangle specified by (x,y,w,h). theta specifies the starting angle and alpha is the extent.

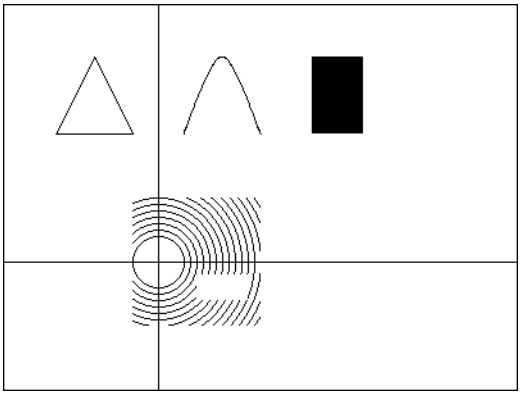
FillPolygon(), FillRectangle(), FillCircle(), and FillArc() are similar to their Draw counterparts, but they fill in the interior of a figure as well as its outline.

EraseArea(x, y, w, h) fills a rectangular area using the background color instead of the foreground color.

An Example

The following code starts by drawing some simple figures. It then moves the origin, draws the new coordinate axes, and sets a clipping region. Circles are drawn around the new origin to show the effect of clipping, then partially erased. The WDone() procedure called at the end waits for the q key to be pressed.

```
WOpen("size=400,300") | stop("can't open window")
DrawPolygon(40, 100, 70, 40, 100, 100)
DrawCurve(140, 100, 170, 40, 200, 100)
FillRectangle(240, 40, 40, 60)
WAttrib("dx=120", "dy=200")
DrawSegment(0, –1000, 0, 1000, 1000, 0, –1000, 0)
Clip(–20, –50, 100, 100)
every DrawCircle(0, 0, 20 to 100 by 5)
EraseArea(30, 10, 40, 20)
WDone()
```



Drawing Attributes

Functions that draw lines are affected by the linewidth and linestyle attributes. The linewidth attribute specifies the thickness of drawn lines. The linestyle attribute can be solid, dashed, or

striped. A value of dashed causes lines to be drawn with regular gaps. A value of striped causes these gaps to be filled with the background color.

The fillstyle and pattern attributes, discussed later, affect all drawing and filling functions.

The drawop attribute specifies the way in which drawn pixels interact with existing pixels. Normally, with drawop=copy, new pixels simply replace existing pixels. When the drawing operation is reverse, new pixels combine with old in such a way as to turn foreground-colored pixels into the background color, and vice versa; the effect on old pixels of any other color is unpredictable.

Text

WWrite(), WWrites(), WRead(), and WReads() are analogous to write(), writes(), read(), and reads(), treating the window as a simple video terminal that scrolls when the bottom is reached.

Output appears at the cursor location, which is defined by row and col attributes or equivalently by x and y attributes. The cursor is moved by GotoRC(row, col) or GotoXY(x, y). The visibility of the cursor is controlled by the cursor attribute.

Characters read by WRead() or WReads() are echoed to the screen at the cursor location if the echo attribute is set. When a character is written or echoed using the cursor, the area behind the character is filled with the background color.

DrawString(x, y, s) outputs text string s without affecting the text cursor position or drawing the background. The first character appears at location (x,y). CenterString(), LeftString(), and RightString() output a string using different positionings.

TextWidth(s) returns the width of a string, measured in pixels.

Fonts

Font(s) sets the text font. An Icon font specification is a comma-separated string giving the family name, style characteristics, and size (a height measured in pixels). Examples are "Helvetica,bold,18" and "Times,bold,italic,14".

Font names and characteristics are system-dependent. However, four portable family names are always available:

mono	a monospaced, sans-serif font
typewriter	a monospaced, serif font
sans	a proportionally spaced, sans-serif font
serif	a proportionally spaced, serif font

The actual fonts vary from one system to another; some systems may not be able to supply fonts with all the correct attributes.

Font specifications are case-insensitive. As an alternative to an Icon font specification, a system-dependent font name can be supplied.

Information about a font can be obtained from the following attributes:

fheight	font height
fwidth	font width
ascent	extent of the font above the baseline

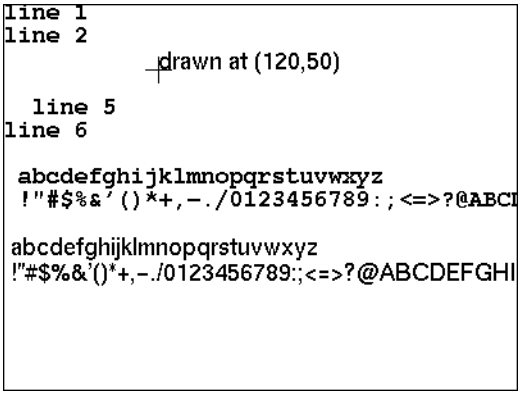
descent	extent of the font below the baseline
leading	distance between baselines of successive text lines

Only the leading attribute can be altered.

An Example

The following example illustrates several aspects of text output. Note the use of newlines and leading spaces in strings in addition to more explicit positioning.

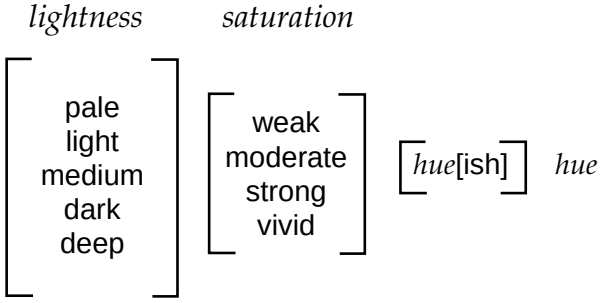
```
WOpen("size=400,300") | stop("can't open window")
Font("typewriter,bold,18")
WWrite("line 1")
WWrite("line 2")
GotoRC(5, 3)
WWrite("line 5")
WWrite("line 6")
WWrite("\n ", &lcase, &ascii)
Font("Helvetica,18")
WWrite("\n ", &lcase, &ascii)
DrawSegment(110, 50, 130, 50, 120, 40, 120, 60)
DrawString(120, 50, "drawn at (120,50)")
WDone()
```



Colors

Fg(s) and Bg(s) set the foreground and background color respectively.

Colors are named by English phrases using a system loosely based on [5]. Examples are "brown", "yellowish green", and "moderate purple–gray". The syntax of a color name is



where choices enclosed in brackets are optional and *hue* can be one of black, gray, white, pink, violet, brown, red, orange, yellow, green, cyan, blue, purple, or magenta. A single hyphen or space separates each word from its neighbor. Color names are insensitive to case; purple and Purple are equivalent.

Conventional English spelling is used. When adding ish to a hue ending in e, the e is dropped. For example, purple becomes purplish. The ish form of red is reddish.

When two hues are supplied and the first hue has no ish suffix, the resulting hue is halfway between the two named hues. When a hue with an ish suffix precedes a hue, the resulting hue is

three-fourths of the way from the ish hue to the main hue. The default lightness is medium and the default saturation is vivid.

Colors can also be specified in terms of red, green, and blue brightness components. The decimal form uses three comma-separated values ranging from 0 to 65535, as in "12000,0,65535". The hexadecimal forms are "#*rgb*", "#*rrggbb*", "#*rrrgggbbb*", and "#*rrrrggggbbbb*", with the longer forms providing greater precision.

Color specifications not recognized by Icon are passed to the graphics system for interpretation, allowing the use of system-dependent names.

ColorValue(s) translates a color specification into decimal form.

Color Correction

Icon colors use a linear scale: the 50% values in "32767,32767,32767" specify a medium gray. Real graphics hardware is nonlinear. When the underlying graphics system does not correct for this, Icon applies its own *gamma correction*. The gamma attribute controls the amount of such correction. A value of 1.0 provides no correction; values between 2 and 3 are appropriate for most uncorrected monitors.

Mutable Colors

On systems with changeable color maps, Icon supports color map access through *mutable colors*.

NewColor(s) reserves a color map entry and returns a negative integer n, a mutable color representing that entry. If s is supplied, the entry is initialized to that color.

An integer returned by NewColor() can be used as a color specification. For example, Fg(n) makes a mutable color the foreground color.

Color(n, s) sets the color map entry of n to the color s. This changes the appearance of any pixels of color n already drawn as well as affecting those drawn subsequently.

FreeColor(n) frees a color map entry when no longer needed, and can be used with normal color specifications as well as mutable colors.

Color Palettes

Color palettes are fixed sets of colors (or grays) used for drawing or reading images. Icon's color palettes are described in Appendix F.

PaletteKey(palette, color) returns a character from the given palette representing an entry in the palette that is close to the given color.

PaletteColor(palette, s) returns the color represented by the single character s in the given palette, or fails if the character is not a member of the palette. The color is returned in the same form as produced by ColorValue().

PaletteChars(palette) returns a string containing the characters that are valid in the given palette. If fails if the palette name is invalid. PaletteGrays(palette) returns only the characters corresponding to shades of gray, ordered from black to white.

Drawing Images

DrawImage(x, y, spec) draws an arbitrarily complex figure in a rectangular area by giving a value to each pixel in the area. x and y specify the upper left corner of the area. spec is a string of the form "*width*,*palette*,*data*" where *width* gives the width of the area to be drawn, *palette* chooses the set of colors to be used, and *data* specifies the pixel values.

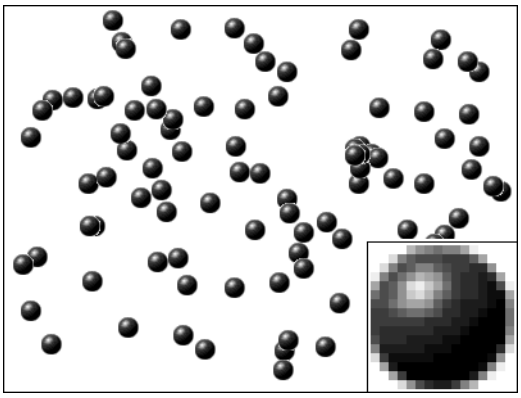
Each character of *data* corresponds to one pixel in the output image. Pixels are written a row at a time, left to right, top to bottom. The amount of data determines the height of the area drawn. The area is always rectangular; the length of the data must be an integral multiple of the width.

The data characters are interpreted in paint-by-number fashion according to the selected palette. Spaces and commas can be used as punctuation to aid readability. The characters ~ and \377 specify transparent pixels that do not overwrite the pixels on the canvas when the image is drawn. Punctuation and transparency characters lose their special meanings in palettes in which they represent colors.

An Example

The following example uses DrawImage() to draw spheres randomly. Transparent pixels are used for better appearance where the spheres overlap. The inset shows a magnified version of a single sphere.

```
WOpen("size=400,300") | stop("can't open window")
sphere := "16,g16, ~~~~B98788AE~~~~_
  ~~D865554446A~~~  ~D856886544339~~ _
  E8579BA9643323A~  A569DECA7433215E_
  7569CDB86433211A  5579AA9643222108_
  4456776533221007   4444443332210007_
  4333333222100008   533322221100000A _
  822222111000003D   D41111100000019~_
  ~A200000000018E~   ~~A4000000028E~~_
  ~~~D9532248B~~~~"
every 1 to 100 do
  DrawImage(?380, ?280, sphere)
WDone()
```



Bi-Level Images

DrawImage() accepts an alternative specification form "*width*,*#data*" for images composed of only the foreground and background colors. The data field is a series of hexadecimal digits specifying row values from top to bottom. Each row is specified by width/4 digits, with fractional values rounded up. An example of a 4-by-4 specification is "4,#9BD9".

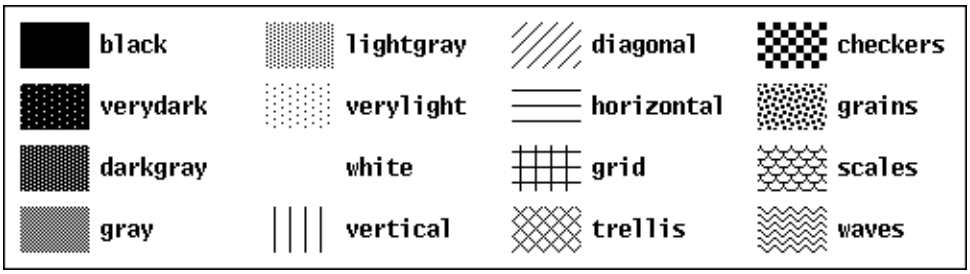
The digits of each row are interpreted as a base-16 number. Each bit of this number corresponds to one pixel; a value of 0 selects the background color and a value of 1 selects the foreground color. The least significant bit corresponds to the left-most pixel.

If the data field is preceded by the character ~ instead of #, the image is written transparently: Bit values of 0 preserve existing pixels instead of writing the background color.

Patterns

The fillstyle attribute, normally solid, can be changed to cause the drawing operations to fill areas or draw lines using a pattern. If the fill style is textured, both the foreground and background colors are used. If the fill style is masked, pixels not set to the foreground color are left unchanged.

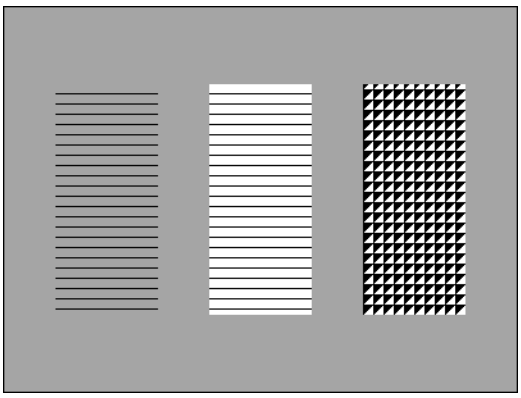
Pattern(spec) sets the pattern attribute, the pattern to be used when the fill style is not solid. spec is a bi-level image specification of the type used with DrawImage() or one of the following predefined names:



An Example

The following example draws three rectangles using patterns. The first two, using the horizontal pattern, differ in fill style. The third uses a custom pattern given as a bi-level specification.

```
WOpen("size=400,300") | stop("can't open window")
Fg("light gray")
FillRectangle()
WAttrib("fg=black", "fillstyle=masked")
Pattern("horizontal")
FillRectangle(40, 60, 80, 180)
WAttrib("fillstyle=textured")
FillRectangle(160, 60, 80, 180)
Pattern("8,#FF7F3F1F0F070301")
FillRectangle(280, 60, 80, 180)
WDone()
```



Miscellaneous Operations

Alert() produces a beep or other signal to attract attention.

CopyArea(x1, y1, w, h, x2, y2) copies the rectangular area (x1,y1,w,h) to (x2,y2). Copying from one window to another is possible by explicitly specifying two window arguments: CopyArea(W1, W2, x1, y1, w, h, x2, y2).

Pixel(x, y, w, h) generates the colors of the pixels in a rectangle, left to right, top to bottom.

ReadImage(s, x, y, p) displays an image from the file named s at (x,y). If p is supplied, the image is displayed using only the colors of palette p. WriteImage(s, x, y, w, h) writes a rectangular area to the file named s. Icon supports GIF format on all platforms; additional formats are also available on some platforms.

WDefault(program, option) returns a custom value registered for the option named option of the program named program.

Events

User actions are passed to an Icon program as *events*. These events do not interrupt execution; they are placed in a queue for retrieval. Events are generated by pressing a key (except modifier keys like the shift key), by resizing the window, by pressing or releasing a button on the mouse, or by moving the mouse while a button is pressed. User resizing of a window is allowed only if the resize attribute is set to "on".

Normal keyboard characters are encoded as one-character strings. Other keys such as the Home key produce integer codes; see Appendix D.

The other events produce integer codes corresponding to keywords:

&lpress	left mouse press
&ldrag	left mouse movement
&lrelease	left mouse release
&mpress	middle mouse press
&mdrag	middle mouse movement
&mrelease	middle mouse release
&rpress	right mouse press
&rdrag	right mouse movement
&rrelease	right mouse release
&resize	window resize

An event is accepted by calling Event(), which returns the code for the next unprocessed event. If no event is available, Event() waits for one to occur.

When Event() returns an event, it also sets the values of keywords that give further information about the event:

&x, &y	mouse location in pixels
&row, &col	mouse location in characters
&control	succeeds if the Control key was pressed
&meta	succeeds if the Meta key was pressed
&shift	succeeds if the Shift key was pressed
&interval	time in milliseconds since previous event

Enqueue(a, x, y, s, i) places event code a in the queue with x and y as the associated mouse coordinates. s is a string containing any of the characters c, m, or s to indicate modifier keys; i gives the interval.

The event queue is an Icon list; Appendix E describes its format. Pending() returns the event list for direct access from Icon. The expression *Pending() returns the size of the list and can be used to check whether an event is available.

Dialogs

Several library procedures display items in a window and then wait for input.

Notice(line1, line2, ...) displays one or more lines of text and then waits for the user to click an Okay button.

OpenDialog(caption, filename) and SaveDialog(caption, filename) each request a text string, normally a file name; they differ in their sets of accompanying buttons. The resulting file name is stored in the global variable dialog_value.

TextDialog() constructs a dialog containing arbitrary numbers of caption lines, text-entry fields, and buttons. SelectDialog() requests a choice from a list of items; ToggleDialog() displays a set of independently selectable buttons. ColorDialog() displays a color-selection window. See Appendix A for details.

The VIB program [3] supports interactive construction of dialog boxes with even more generality. It allows arbitrary placement of buttons and text within a dialog box and provides additional devices such as sliders and scroll bars.

Windows

WOpen() opens a window and returns a value of type window. If &window is null, WOpen() assigns the newly opened window to it. Attribute values can be given as arguments to WOpen() to configure the new window.

WFlush() flushes the window's output buffer, forcing the immediate display of any output that has been buffered. WSync() flushes the buffer and does not return until all pending output has been displayed. WDelay(i) flushes the buffer and then delays i milliseconds. WClose() closes a window.

The size and pos attributes (and other related attributes) can be used to resize or reposition a window. The resize attribute controls whether the user is allowed to resize the window. Raise() and Lower() adjust the window stacking order on the screen.

The canvas attribute specifies window visibility. The default value is normal. With canvas=maximal, the window fills the screen. With canvas=hidden, the window is not visible on the screen. With canvas=iconic, a minimized icon or label is displayed. The attributes iconlabel, iconimage, and iconpos can affect the appearance of the window while in this state.

Active() lets a program multiplex input from several windows. It checks all open windows and returns a window for which an event is available; it waits if there are no unprocessed events.

Graphics Contexts

A window is composed of two parts: a canvas, the visible or invisible area used for drawing, and a graphics context, a set of parameters that control drawing.

All the attributes of a window are associated with either its canvas or its graphics context. The tables in Appendix C list the attributes by category.

Clone() creates a new graphics context coupled with an existing canvas. The resulting window value shares the canvas with an existing window but has an independent set of graphics attributes. These attributes are initialized to the same values as in the original window, then modified by any arguments passed to Clone().

Couple(W1, W2) produces a window that couples the canvas of W1 with the graphics attributes of W2. This allows a set of graphics context attributes to be shared across multiple canvases.

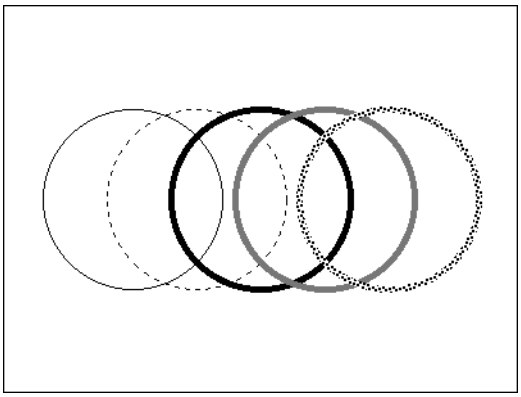
Uncouple() discards a window value. If there are no other references to the window's canvas, it disappears from the screen.

An Example

This example illustrates the use of graphics contexts. First, several windows are created by cloning, each with different attributes. Note that the two clones of wide inherit its linewidth attribute. Then, the cloned windows are used to draw circles, showing the effects of the different attributes.

```
WOpen("size=400,300") | stop("can't open window")
dashed := Clone("linestyle=dashed")
wide := Clone("linewidth=5")
gray := Clone(wide, "fg=gray")
patt := Clone(wide, "fillstyle=textured", "pattern=grains")

DrawCircle(100, 150, 70)
DrawCircle(dashed, 150, 150, 70)
DrawCircle(wide, 200, 150, 70)
DrawCircle(gray, 250, 150, 70)
DrawCircle(patt, 300, 150, 70)
WDone()
```



Further Information

Appendix A provides detailed documentation of all built-in graphics functions and all library procedures mentioned here. Additional library procedures are described in [4].

Some of the programs in the library can be run to help learn about color in Icon. The colrbook and colrpick programs allow interactive exploration of English and numeric color specifications, respectively. The palette program displays any of the predefined color palettes.

Many other library procedures make extensive use of the graphics facilities. Much can be learned by browsing through the source code.

Acknowledgments

Many people have assisted the development of Icon's graphics capabilities. Darren Merrill, Ken Walker, Nick Kline, and Jon Lipp contributed to the design. Darren Merrill, Sandy Miller, and Cheyenne Wills assisted with aspects of the implementation. Steve Wampler and Bob Alexander provided suggestions, bug reports, and program examples.

References

1. R. E. Griswold and M. T. Griswold, *The Icon Programming Language*, Peer-to-Peer Communications, Inc., San Jose, CA, third edition, 1996.

2. R. E. Griswold, C. L. Jeffery, and G. M. Townsend, *Version 9.3 of the Icon Programming Language*, The Univ. of Arizona Icon Project Document IPD278, 1996.

3. G. M. Townsend and M. Cameron, *VIB: A Visual Interface Builder for Icon*, The Univ. of Arizona Icon Project Document IPD265, 1995.

4. R. E. Griswold, *The Icon Program Library; Version 9.3*, The Univ. of Arizona Icon Project Document IPD279, 1996.

5. Berk T., Brownstein L., and Kaufman A., "A New Color-Naming System for Graphics Languages", *IEEE Computer Graphics & Applications*, May 1982, 37-44.

Appendix A

Graphics Procedures

In addition to the notation used in the Icon language book [1] to denote argument types, the following characters have meaning as indicated:

W	window	graphics source or destination
a	any type	arbitrary value
x, y	integer	coordinate location
w, h	integer	width and height of a rectangle
theta	real	angle (measured in radians)
alpha	real	angle (measured in radians)
k	string or integer	color specification

Either or both of w and h can be negative to indicate a rectangle that extends leftward or upward from its given coordinates. A color specification is either an integer obtained from NewColor() or a string having one of these forms:

*[lightness] [saturation] [hue[*ish*]] hue*
red,green,blue
#hexdigits
system-dependent-color-name

Any window argument named W can be omitted, in which case the subject window, &window, is used. Note that this is not the same as a default argument: to use the subject window, the argument is omitted entirely, not replaced by a null argument.

The notation "**……**" in an argument list indicates that additional argument sets can be provided, producing the same effect as multiple calls. The optional window argument, W, is not repeated in these additional argument sets.

The list that follows includes some procedures that are not built into Icon itself but are instead part of the library. For these, the corresponding link file is noted. Alternatively, link graphics incorporates all procedures listed.

Only a small portion of the library is documented here; the full library [4] is much more extensive.

Active() : W — *produce active window*

Active() returns a window that has one or more events pending, waiting if necessary. Successive calls avoid window starvation by checking the open windows in a different order each time. Active() fails if no window is open.

See also: Pending()

Alert(W) : W — *alert user*

Alert() produces a beep or other signal to attract attention.

Bg(W, k1) : k2 — *set or query background color*

Bg() returns the background color. If k1 is supplied, the color is first set to that specification; failure occurs if the request cannot be satisfied. Setting the background color does not change the appearance of the window, but subsequent drawing operations that use the background color are affected.

See also: EraseArea(), Fg(), and FreeColor()

CenterString(W, x, y, s) : W — *draw centered string*

CenterString() draws a text string that is centered vertically and horizontally about (x,y).

Link: gpxop

See also: DrawString(), LeftString(), and RightString()

Clip(W, x, y, w, h) : W — *set clipping rectangle*

Clip() sets the clipping region to the specified rectangle; subsequent output extending outside its bounds is discarded. If Clip() is called with no arguments, clipping is disabled and the entire canvas is writable.

Defaults: w, h to edge of window

Clone(W, s1, s2, …, sn) : W — *create new context with existing canvas*

Clone() produces a new window value that combines the canvas of W with a new graphics context. The new graphics attributes are copied from W and modified by the arguments of Clone(). Invalid arguments produce failure as in WAttrib().

See also: Couple() and WAttrib()

Color(W, i, k1, ……) : k2 — *set or query mutable color*

Color() returns the setting of mutable color i if k1 is omitted. If k1 is supplied, color i is changed as specified, with an immediate effect on any visible pixels of that color. Additional index and color pairs may be supplied to set multiple entries with one call. Color() fails if a color specification is invalid.

See also: NewColor()

ColorDialog(W, L, k, p, a) : s — *display color selection dialog*

ColorDialog() displays a color selection dialog box with Okay and Cancel buttons. The box is headed by zero or more captions specified by the list L, or a single string argument if passed in place of a list. If k is supplied, it specifies a reference color to be displayed below the color being adjusted.

If a callback procedure p is supplied, then p(a, s) is called whenever the color settings are adjusted. The argument a is an arbitrary value from the ColorDialog() call; s is the new color setting in the form returned by ColorValue().

The color initially is set to k , if supplied, or otherwise to the foreground color.

The final color setting, in ColorValue() form, is stored in the global variable dialog_value. ColorDialog() returns the name of the button that was selected.

Defaults: L "Select color:"

Link: dialog

ColorValue(W, k) : s — *translate color to canonical form*

ColorValue() interprets the color k and returns a string of three comma-separated integer values denoting the color's red, green, and blue components. ColorValue() fails if k is not a valid color specification.

CopyArea(W1, W2, x1, y1, w, h, x2, y2) : W1 — *copy rectangle*

CopyArea() copies a rectangular region (x1, y1, w, h) of window W1 to location (x2, y2) on window W2. If W2 is omitted, W1 is used as both source and destination. If W1 is omitted, the subject window is used.

Defaults:		
	x1, y1	upper-left pixel
	w, h	to edge of window
	x2, y2	upper-left pixel

Couple(W1, W2) : W3 — *couple canvas and context*

Couple() produces a new window value that binds the canvas of W1 with the graphics context of W2. Both arguments are required.

See also: Clone() and WAttrib()

DrawArc(W, x, y, w, h, theta, alpha, ……) : W — *draw arc*

DrawArc() draws an arc of the ellipse inscribed in the rectangle specified by (x, y, w, h). The arc begins at angle theta and extends by an angle alpha.

Defaults:		
	x, y	upper-left pixel
	w, h	to edge of window
	theta	0
	alpha	2π

See also: DrawCircle() and FillArc()

DrawCircle(W, x, y, r, theta, alpha, ……) : W — *draw circle*

DrawCircle() draws an arc or circle of radius r centered at (x,y). theta is the starting angle and alpha is the extent of the arc.

Defaults: theta 0
alpha 2π

See also: DrawArc() and FillCircle()

DrawCurve(W, x1, y1, x2, y2, …, xn, yn) : W — *draw curve*

DrawCurve() draws a smooth curve through the points given as arguments. If the first and last point are the same, the curve is smooth and closed through that point.

See also: DrawLine() and DrawPolygon()

DrawImage(W, x, y, s) : i — *draw rectangular figure*

DrawImage() draws an arbitrarily complex figure in a rectangular area at (x,y). s has one of these forms:

"*width,palette,data*"	character-per-pixel image
"*width,#hexdigits*"	bi-level image
"*width,~hexdigits*"	transparent bi-level image

DrawImage() normally returns the null value, but if some colors cannot be allocated, it returns the number of colors that cannot be allocated.

Defaults: x, y upper-left pixel

See also: Pattern() and ReadImage()

DrawLine(W, x1, y1, x2, y2, …, xn, yn) : W — *draw line*

DrawLine() draws line segments connecting a list of points in succession.

See also: DrawCurve(), DrawPolygon(), and DrawSegment()

DrawPoint(W, x, y, ……) : W — *draw point*

DrawPoint() draws a point at each coordinate location given.

DrawPolygon(W, x1, y1, …, xn, yn) : W — *draw polygon*

DrawPolygon() draws the outline of a polygon formed by connecting the given points in order, with x1,y1 following xn,yn.

See also: DrawCurve(), DrawLine(), and FillPolygon()

DrawRectangle(W, x, y, w, h, ……) : W — *draw rectangle*

DrawRectangle() draws the outline of the rectangle with corners at (x,y) and (x+w,y+h).

Defaults:	x, y	upper-left pixel
	w, h	to edge of window

See also: FillRectangle()

DrawSegment(W, x1, y1, x2, y2, ……) : W — *draw line segment*

DrawSegment() draws a line between two points. Additional pairs of coordinates may be supplied to draw additional, disconnected segments.

See also: DrawLine()

DrawString(W, x, y, s, ……) : W — *draw text*

DrawString() draws a string of characters starting at (x,y) without altering the text cursor.

Enqueue(W, a, x, y, s, i) : W — *append event to queue*

Enqueue() adds event a to the window event list with an event location of (x,y). The string s specifies a set of modifier keys using the letters c, m, and s to represent &control, &meta, and &shift, respectively. i specifies a value for &interval, in milliseconds.

Defaults:	a	&null
	x	0
	y	0
	s	""
	i	0

Link: enqueue

See also: Pending()

EraseArea(W, x, y, w, h, ……) : W — *clear rectangular area*

EraseArea() fills a rectangular area with the background color.

Defaults:	x, y	upper-left pixel
	w, h	to edge of window

See also: FillRectangle()

Event(W) : a — *return next window event*

Event() returns the next event from a window, waiting if necessary. The keywords &x, &y, &row, &col, &interval, &control, &shift, and &meta are set as side effects of calling Event().

See also: Active(), Enqueue(), Pending(), WRead(), and WReads()

Fg(W, k1) : k2 — *set or query foreground color*

Fg() returns the foreground color. If k1 is supplied, the color is first set to that specification; failure occurs if the request cannot be satisfied. Setting the foreground color does not change the appearance of the window, but subsequent drawing operations are affected.

See also: Bg(), FreeColor(), and Shade()

FillArc(W, x, y, w, h, theta, alpha, ……) : W — *draw filled arc*

FillArc() draws a filled arc of the ellipse inscribed in the rectangle specified by (x, y, w, h). The arc begins at angle theta and extends by an angle alpha.

Defaults:		
	x, y	upper-left pixel
	w, h	to edge of window
	theta	0
	alpha	2π

See also: DrawArc() and FillCircle()

FillCircle(W, x, y, r, theta, alpha, ……) : W — *draw filled circle*

FillCircle() draws a filled arc or circle of radius r centered at (x,y). theta is the starting angle and alpha is the extent of the arc.

Defaults:		
	theta	0
	alpha	2π

See also: DrawCircle() and FillArc()

FillPolygon(W, x1, y1, x2, y2, …, xn, yn) : W — *draw filled polygon*

FillPolygon() draws and fills the polygon formed by connecting the given points in order, with x1,y1 following xn,yn.

See also: DrawPolygon()

FillRectangle(W, x, y, w, h, ……) : W — *draw filled rectangle*

FillRectangle() draws a filled rectangle.

Defaults:		
	x, y	upper-left pixel
	w, h	to edge of window

See also: DrawRectangle() and EraseArea()

Font(W, s1) : s2 — *set or query text font*

Font() returns the text font. If s1 is supplied, the font is first set to that specification; failure occurs if the request cannot be satisfied.

FreeColor(W, k, ……) : W — *free color*

FreeColor() informs the graphics system that the color k no longer appears in the window. This may allow the system to reclaim some resources. Unpredictable results may occur if the color is still present in the window.

See also: Bg(), Fg(), and NewColor()

GotoRC(W, i1, i2) : W — *move text cursor to row and column*

GotoRC() sets the text cursor position to row i1 and column i2, where the character position in the upper-left corner of the window is 1,1 and calculations are based on the current font attributes.

Defaults: x, y 1, 1

See also: GotoXY()

GotoXY(W, x, y) : W — *move text cursor to coordinate position*

GotoXY() sets the text cursor position to the specified coordinate position.

Defaults: x, y 0, 0

See also: GotoRC()

LeftString(W, x, y, s) : W — *draw left-justified string*

LeftString() draws a text string that is left-justified at position x and centered vertically about y.

Link: gpxop

See also: CenterString(), DrawString(), and RightString()

Lower(W) : W — *lower window to bottom of window stack*

Lower() sets a window to be "below" all other windows, causing it to become obscured by windows that overlap it.

See also: Raise()

NewColor(W, k) : i — *allocate mutable color*

NewColor() allocates a changeable entry in the color map and returns a small negative integer that serves as a handle to this entry. If k is supplied, the color map entry is initialized to that color. NewColor() fails if no mutable entry is available.

See also: Color() and FreeColor()

Notice(W, s1, s2, …, sn) : sm — *display strings and await response*

Notice() posts a dialog box with an Okay button and returns "Okay" after response by the user. Each string sn is displayed centered on a separate line in the dialog box.

Link: dialog

See also: TextDialog()

OpenDialog(W, s1, s2, i) : s3 — *display dialog for opening file*

OpenDialog() displays a dialog box allowing entry of a text string of up to i characters, normally a file name, along with Okay and Cancel buttons. s1 supplies a caption to be displayed in the dialog box. s2 is used as the initial value of the editable text string. The final text string value is stored in the global variable dialog_value. OpenDialog() returns the name of the button that was selected.

Defaults: s1 "Open:"
s2 ""
i 50

Link: dialog

See also: SaveDialog() and TextDialog()

PaletteChars(W, s1) : s2 — *return characters of color palette*

PaletteChars() returns the string of characters that index the colors of palette s1.

Default: s1 "c1"

See also: PaletteColor(), PaletteGrays(), and PaletteKey()

PaletteColor(W, s1, s2) : s3 — *return color from palette*

PaletteColor() returns the color indexed by character s2 in palette s1. The result is in the form produced by ColorValue().

Default: s1 "c1"

See also: ColorValue(), PaletteChars(), PaletteGrays(), and PaletteKey()

PaletteGrays(W, s1) : s2 — *return grayscale entries of palette*

PaletteGrays() returns the string of characters that index the achromatic entries within palette s1, ordered from black to white.

Link: color

See also: PaletteChars(), PaletteColor(), and PaletteKey()

PaletteKey(W, s1, k) : s2 — *return character of closest color in palette*

PaletteKey() returns the character indexing the color of palette s1 that is closest to the color k.

Default: s1 "c1"

See also: PaletteChars(), PaletteGrays(), and PaletteColor()

Pattern(W, s) : W — *set fill pattern*

Pattern() sets a pattern to be used for drawing when the fill style is set to "masked" or "textured". s can be a known pattern name or a specification of the form "*width,#data*" where the data is given by hexadecimal digits. Pattern() fails in the case of a bad specification or unknown name.

See also: DrawImage()

Pending(W) : L — *produce event list*

Pending() returns the list that holds the pending events of a window. If no events are pending, this list is empty.

See also: Enqueue() and Event()

Pixel(W, x, y, w, h) : k1, k2, …, kn — *generate pixel values*

Pixel() generates the colors of the pixels in the given rectangle, left to right, top to bottom.

Defaults: x, y upper-left pixel
w, h to edge of window

Raise(W) : W — *raise window to top of window stack*

Raise() sets a window to be "above" all other windows so that it is not obscured by any other window.

See also: Lower()

ReadImage(W, s1, x, y, s2) : i — *load image file*

ReadImage() loads an image from file s1, placing its upper-left corner at x,y. If a palette s2 is supplied, the colors of the image are mapped to those of the palette. ReadImage() fails if it cannot read an image from file s1. It normally returns the null value, but if some colors cannot be allocated, it returns the number of colors that cannot be allocated.

Defaults: x, y upper-left pixel

See also: DrawImage() and WriteImage()

RightString(W, x, y, s) : W — *draw right-justified string*

RightString() draws a text string that is right-justified at position x and centered vertically about y.

Link: gpxop

See also: CenterString(), DrawString(), and LeftString()

SaveDialog(W, s1, s2, i) : s3 — *display dialog for saving file*

SaveDialog() displays a dialog box allowing entry of a text string of up to i characters, normally a file name, along with Yes, No, and Cancel buttons. s1 supplies a caption to be displayed in the dialog box. s2 is used as the initial value of the editable text string. The final text string value is stored in the global variable dialog_value. SaveDialog() returns the name of the button that was selected.

Defaults:

s1	"Save:"
s2	""
i	50

Link: dialog

See also: OpenDialog() and TextDialog()

SelectDialog(W, L1, L2, s, L3, i) : s — *display selection dialog*

SelectDialog() constructs and displays a dialog box and waits for the user to select a button. The box contains zero or more captions specified by the list L1, zero or more radio buttons specified by L2 and s, and one or more buttons specified by L3. i specifies the index of the default button, with a value of 0 specifying that there is no default button. Any of the list arguments L*n* can be specified by a single non-null value which is then treated as a one-element list.

For the radio buttons, L2 specifies the button names and s specifies the name for the default button. If L2 is omitted, there are no buttons.

SelectDialog() returns the name of the button that was selected to dismiss the dialog. The global variable dialog_value is assigned the name of the selected radio button.

Defaults:

L1	[]
L2	[]
L3	["Okay", "Cancel"]
i	1

Link: dialog

See also: TextDialog() and ToggleDialog()

Shade(W, k) : W — *set foreground for area filling*

Shade() sets the foreground color to k on a color or grayscale display. On a bi-level display, it sets the fill style to textured and installs a dithering pattern that approximates the brightness of color k.

Link: color

See also: Fg()

TextDialog(W, L1, L2, L3, L4, L5, i) : s — *display text dialog*

TextDialog() constructs and displays a dialog box and waits for the user to select a button. The box contains zero or more captions specified by the list L1, zero or more text-entry fields specified by L2, L3, and L4, and one or more buttons specified by L5. i specifies the index of the default button, with a value of 0 specifying that there is no default button. Any of the list arguments L*n* can be specified by a single non-null value which is then treated as a one-element list.

For the text-entry fields, L2 specifies the labels, L3 specifies the default values, and L4 specifies the maximum widths. If L2, L3, and L4 are not the same length, the shorter lists are extended as necessary by duplicating the last element. If omitted entirely, the defaults are: no labels, no initial values, and a width of 10 (or more if necessary to hold a longer initial value).

TextDialog() returns the name of the button that was selected to dismiss the dialog. The global variable dialog_value is assigned a list containing the name of the text fields.

Defaults:

L1	[]
L2	[]
L3	[]
L4	[]
L5	["Okay", "Cancel"]
i	1

Link: dialog

See also: Notice(), OpenDialog(), SaveDialog(), and SelectDialog()

TextWidth(W, s) : i — *return width of text string*

TextWidth() returns the width of string s, in pixels, as drawn using the current font.

See also: DrawString()

ToggleDialog(W, L1, L2, L3, L4, i) : L — *display toggle dialog*

ToggleDialog() constructs and displays a dialog box and waits for the user to select a button. The box contains zero or more captions specified by the list L1, zero or more toggle buttons specified by L2, zero or more toggle states (1 or null) specified by L3, and one or more buttons specified by L4. i specifies the index of the default button, with a value of 0 specifying that there is no default button. Any of the list arguments L*n* can be specified by a single non-null value, which is then treated as a one-element list.

For the toggle buttons, L2 specifies the labels and L3 specifies the corresponding states. If L2 and L3 are not the same length, the shorter list is extended as necessary by duplicating the last element. If omitted entirely, the defaults are: no labels and null states.

ToggleDialog() returns the name of the button that was selected to dismiss the dialog. The global variable dialog_value is assigned a list containing the states of the toggle buttons.

Defaults:	L1	[]
	L2	[]
	L3	[]
	L4	["Okay", "Cancel"]
	i	1

Link: dialog

See also: SelectDialog() and TextDialog()

Uncouple(W) : W — *uncouple window*

Uncouple() frees the window W. If no other bindings to the same canvas exist, the window is closed.

See also: Clone(), Couple(), and WClose()

WAttrib(W, s1, s2, …, sn) : a1, a2, …, an — *set or query attributes*

WAttrib() sets and generates window attribute values. Each string of the form *name=value* sets a value; a string with just a name is an inquiry. First, any requested values are set. Then WAttrib() generates the values of all referenced attributes. Each value has the data type appropriate to the attribute it represents. WAttrib() ignores unrecognized names and illegal values, producing no result; if all arguments are invalid, WAttrib() fails.

WClose(W) : W — *close window*

WClose() closes a window. The window disappears from the screen, and all bindings of its canvas are rendered invalid. Closing the subject window sets &window to the null value.

Link: wopen

See also: Uncouple(), WFlush(), and WOpen()

WDefault(W, s1, s2) : s3 — *get default value from environment*

WDefault() returns the value of option s2 for the program named s1 as registered with the graphics system. If no such value is available, or if the system provides no registry, WDefault() fails.

WDelay(W, i) : W — *flush window and delay*

WDelay() flushes any pending output for window W and then delays for i milliseconds before returning.

Default: i 1

Link: wopen

See also: WFlush()

WDone(W) — *wait for "quit" event, then exit*

WDone() waits until a q or Q is entered, then terminates program execution. It does not return.

Link: wopen

See also: WQuit()

WFlush(W) : W — *flush pending output to window*

WFlush() forces the execution of any window commands that have been buffered internally and not yet executed.

See also: WClose(), WDelay(), and WSync()

WOpen(s1, s2, …, sn) : W — *open and return window*

WOpen() creates and returns a new window having the attributes specified by the argument list. Invalid arguments produce failure or error as in WAttrib(). If &window is null, the new window is assigned as the subject window.

Link: wopen

See also: WAttrib() and WClose()

WQuit(W) : W — *check for "quit" event*

WQuit() consumes events until a q or Q is entered, at which point it returns. If the event queue is exhausted first, WQuit() fails.

Link: wopen

See also: WDone()

WRead(W) : s — *read line from window*

WRead() accumulates characters typed in a window until a newline or return is entered, then returns the resulting string (without the newline or return). Backspace and delete characters may be used for editing. The typed characters are displayed in the window if the echo attribute is set.

Link: wopen

See also: Event() and WReads()

WReads(W, i) : s — *read characters from window*

WReads() returns the next i characters typed in a window. Backspace and delete characters may be used for editing prior to entry of character i. The typed characters are displayed in the window if the echo attribute is set.

Default: i 1

Link: wopen

See also: Event() and WRead()

WriteImage(W, s, x, y, w, h) : W — *write image to file*

WriteImage() writes an image of the rectangular area (x,y,w,h) to the file s. It fails if s cannot be written or if the specified area, after clipping by the window's edges, has a width or height of zero. The file is normally written in GIF format, but some forms of file names may select different formats on some graphics systems.

Defaults: x, y upper-left pixel
w, h to edge of window

See also: ReadImage()

WSync(W) : W — *synchronize with server*

WSync() synchronizes the program with the graphics server on a client-server graphics system, returning after all pending output has been processed. On systems that maintain synchronization at all times, WSync() has no effect.

See also: WFlush()

WWrite(W, s1, s2, …, sn) : sn — *write line to window*

WWrite() writes a string to a window at the text cursor position. The area behind the written text is set to the background color. Newline, return, tab, backspace, and delete characters reposition the cursor. An implicit newline is output following the last argument.

Link: wopen

See also: DrawString() and WWrites()

WWrites(W, s1, s2, …, sn) : sn — *write partial line to window*

WWrite() writes a string to a window at the text cursor position. The area behind the written text is set to the background color. Newline, return, tab, backspace, and delete characters reposition the cursor. Unlike WWrite(), no newline is added.

Link: wopen

See also: DrawString() and WWrite()

Appendix B

Graphics Keywords

&col : i — *mouse column*

The value of &col is normally the column location of the mouse at the time of the last received window event. If a window is open, &col may also be changed by assignment, which also affects &x, or as a side effect of assignment to &x.

&control : n — *state of control key during window event*

The value of &control is the null value if the control key was depressed at the time of the last received window event; otherwise, a reference to &control fails.

&interval : i — *elapsed time between window events*

The value of &interval is the time, in milliseconds, between the last received window event and the previous event in that window. &interval is zero if this information is not available.

&ldrag : i — *left-button drag event*

The value of &ldrag is the integer that represents the event of dragging the mouse with the left button depressed.

&lpress : i — *left-button press event*

The value of &lpress is the integer that represents the event of pressing the left mouse button.

&lrelease : i — *left-button release event*

The value of &lrelease is the integer that represents the event of releasing the left mouse button.

&mdrag : i — *middle-button drag event*

The value of &mdrag is the integer that represents the event of dragging the mouse with the middle button depressed.

&meta : n — *state of meta key during window event*

The value of &meta is the null value if the meta key was depressed at the time of the last received window event; otherwise, a reference to &meta fails.

&mpress : i — *middle-button press event*

The value of &mpress is the integer that represents the event of pressing the middle mouse button.

&mrelease : i — *middle-button release event*

The value of &mrelease is the integer that represents the event of releasing the middle mouse button.

&rdrag : i — *right-button drag event*

The value of &rdrag is the integer that represents the event of dragging the mouse with the right button depressed.

&resize : i — *window resize event*

The value of &resize is the integer that represents a window resizing event.

&row : i — *mouse row location*

The value of &row is normally the column location of the mouse at the time of the last received window event. If a window is open, &row may also be changed by assignment, which also affects &y, or as a side effect of assignment to &y.

&rpress : i — *right-button press event*

The value of &rpress is the integer that represents the event of pressing the right mouse button.

&rrelease : i — *right-button release event*

The value of &rrelease is the integer that represents the event of releasing the right mouse button.

&shift : n — *state of shift key during window event*

The value of &shift is the null value if the shift key was depressed at the time of the last received window event; otherwise, a reference to &shift fails.

&window : W — *subject window*

The value of &window is the subject window, the default window for most graphics procedures. It may be changed by assignment. If there is no subject window, &window is null.

&x : i — *mouse x-coordinate*

The value of &x is normally the x-coordinate of the mouse at the time of the last received window event. If a window is open, &x may also be changed by assignment, which also affects &col, or as a side effect of assignment to &col.

&y : i — *mouse y-coordinate*

The value of &y is normally the y-coordinate of the mouse at the time of the last received window event. If a window is open, &y may also be changed by assignment, which also affects &row, or as a side effect of assignment to &row.

Appendix C

Window Attributes

Window attributes describe and control various characteristics of a window. Some attributes are fixed and can only be read; others can be set only when the window is opened. Most can be changed at any time.

There are two classes of attributes: *canvas attributes* and *graphics context attributes*. In general, canvas attributes relate to aspects of the window itself, while graphics context attributes affect drawing operations. Alternate graphics contexts, each with its own set of graphics context attributes, are created by Clone(). Canvas attributes, however, are shared by all clones of a window.

Initial attribute settings can be passed as arguments to WOpen() or Window(). For an existing window, attributes are read or written by calling WAttrib(). Specific procedures also exist for reading or writing certain attributes; these are noted in the *See also* sections of the individual attribute descriptions.

In the tables that follow, the letter R indicates attributes that can be read by WAttrib() and the letter W indicates attributes that can be written — either initially or by calling WAttrib(). Writable graphics context attributes can also be set by Clone().

Canvas Attributes

The following attributes are associated with a canvas and shared by all windows that reference that canvas.

Usage	Canvas Attribute	Interpretation
R, W	label	window label (title)
R, W	pos, posx, posy	window position on screen
R, W	resize	user resizing flag
R, W	size, height, width	window size, in pixels
R, W	lines, columns	window size, in characters
W	image	initial canvas contents
R, W	canvas	window visibility state
W	iconpos	icon position
R, W	iconlabel	icon label
R, W	iconimage	icon image
R, W	echo	character echoing flag
R, W	cursor	text cursor visibility flag
R, W	x, y	cursor location, in pixels
R, W	row, col	cursor location, in characters
R, W	pointer	pointer (mouse) shape
R, W	pointerx, pointery	pointer location, in pixels
R, W	pointerrow, pointercol	pointer location, in characters
R, W	display	device on which the window appears
R	depth	display depth, in bits
R	displayheight	display height, in pixels
R	displaywidth	display width, in pixels

Graphics Context Attributes

The following attributes are associated with a graphics context.

Usage	Graphics Attribute	Interpretation
R, W	fg	foreground color
R, W	bg	background color
R, W	reverse	color reversal flag
R, W	drawop	drawing operation
R, W	gamma	color correction factor
R, W	font	text font
R	fheight, fwidth	maximum character size
R	ascent, descent	dimensions from baseline
R, W	leading	vertical advancement
R, W	linewidth	line width
R, W	linestyle	line style
R, W	fillstyle	fill style
R, W	pattern	fill pattern
R, W	clipx, clipy	clipping rectangle position
R, W	clipw, cliph	clipping rectangle extent
R, W	dx, dy	output translation

Attribute Descriptions

---

ascent — *text font ascent*

The read-only graphics context attribute ascent gives the distance, in pixels, that the current text font extends above the baseline.

See also: descent and fheight

---

bg — *background color*

The graphics context attribute bg specifies current background color.

Initial value: "white"

See also: fg, drawop, gamma, reverse, and Bg()

---

canvas — *window visibility*

The canvas attribute canvas specifies the window visibility.

Values: "hidden", "iconic", "normal", "maximal"

Initial value: "normal"

See also: Lower() and Raise()

---

cliph — *height of clipping region*

The graphics context attribute cliph specifies the height of the clipping region.

Initial value: &null (clipping disabled)

See also: clipw, clipx, clipy, and Clip()

---

clipw — *width of clipping region*

The graphics context attribute clipw specifies the width of the clipping region.

Initial value: &null (clipping disabled)

See also: cliph, clipx, clipy, and Clip()

---

clipx — *x-coordinate of clipping region*

The graphics context attribute clipx specifies the left edge of the clipping region.

Initial value: &null (clipping disabled)

See also: cliph, clipw, clipy, and Clip()

clipy — *y-coordinate of clipping region*

The graphics context attribute clipy specifies the top edge of the clipping region.

Initial value: &null (clipping disabled)

See also: cliph, clipw, clipx, and Clip()

---

col — *text cursor column*

The canvas attribute col specifies the horizontal position of the text cursor, measured in characters.

See also: cursor, row, x, and y

---

columns — *window width in characters*

The graphics context attribute columns specifies the number of text columns available using the current font.

Initial value: 80

See also: lines and width

---

cursor — *text cursor visibility flag*

The canvas attribute cursor specifies whether the text cursor is actually visible on the screen. The text cursor appears only when the program is blocked waiting for input.

Values: "on", "off"

Initial value: "off"

See also: col, echo, row, x, and y

---

depth — *number of bits per pixel*

The read-only canvas attribute depth gives the number of bits allocated to each pixel by the graphics system.

---

descent — *text font descent*

The read-only graphics context attribute descent gives the distance, in pixels, that the current text font extends below the baseline.

See also: ascent and fheight

---

display — *name of display screen*

The canvas attribute display specifies the particular monitor on which the window appears. It cannot be changed after the window is opened.

displayheight — *height of display screen*

The read-only canvas attribute displayheight gives the height in pixels of the display screen on which the window is placed.

See also: displaywidth

---

displaywidth — *width of display screen*

The read-only canvas attribute displaywidth gives the width in pixels of the display screen on which the window is placed.

See also: displayheight

---

drawop — *drawing mode*

The graphics context attribute drawop specifies the way in which newly drawn pixels are combined with the pixels that are already in a window.

Values: "copy", "reverse"

Initial value: "copy"

See also: bg, fg, and reverse

---

dx — *horizontal translation*

The graphics context attribute dx specifies a horizontal offset that is added to the x value of every coordinate pair before interpretation.

Initial value: 0

See also: dy

---

dy — *vertical translation*

The graphics context attribute dy specifies a vertical offset that is added to the y value of every coordinate pair before interpretation.

Initial value: 0

See also: dx

---

echo — *character echoing flag*

The canvas attribute echo specifies whether keyboard characters read by WRead() and WReads() are echoed in the window. When echoing is enabled, the characters are echoed at the text cursor position.

Values: "on", "off"

Initial value: "on"

See also: cursor, WRead(), and WReads()

fg — *foreground color*

The graphics context attribute fg specifies the current foreground color.

Initial value: "black"

See also: bg, drawop, gamma, reverse, and Fg()

---

fheight — *text font height*

The read-only graphics context attribute fheight gives the overall height of the current text font.

See also: ascent, descent, fwidth, and leading

---

fillstyle — *area filling style*

The graphics context attribute fillstyle specifies whether a pattern is to be used when drawing. The fill style affects lines and text as well as solid figures. The pattern itself is set by the pattern attribute.

Values: "solid", "textured", "masked"

Initial value: "solid"

See also: linestyle and pattern

---

font — *text font name*

The graphics context attribute font specifies the current text font.

Initial value: "fixed"

See also: Font()

---

fwidth — *text font width*

The read-only graphics context attribute fwidth gives the width of the widest character of the current text font.

See also: fheight

---

gamma — *color correction factor*

The graphics context attribute gamma specifies the amount of color correction applied when converting between Icon color specifications and those of the underlying graphics system. A value of 1.0 results in no color correction. Larger values produce lighter, less saturated colors.

Values: real values greater than zero

Initial value: system dependent

See also: fg and bg

height — *window height in pixels*

The canvas attribute height specifies the height of the window.

Initial value: enough for 12 lines of text

See also: lines, size, and width

---

iconimage — *window image when iconified*

The canvas attribute iconimage names a file containing an image to be used as the representation of the window when iconified.

Initial value: none

See also: iconlabel, iconpos, and image

---

iconlabel — *window label when iconified*

The canvas attribute iconlabel specifies a label to be used as the representation of the window when iconified.

Initial value: initial value of label attribute

See also: iconimage, iconpos, and label

---

iconpos — *window position when iconified*

The write-only canvas attribute iconpos specifies the location of the iconified window as a string containing comma-separated x- and y-coordinates.

See also: iconimage and iconlabel

---

image — *source of window contents*

The write-only canvas attribute image names a file containing an image to be used as the initial contents of a window when it is opened.

See also: iconimage

---

label — *window label*

The canvas attribute label specifies a title used to identify the window.

Initial value: ""

See also: iconlabel

leading — *text line advancement*

The graphics context attribute leading specifies the vertical spacing of successive lines of text written in a window.

Initial value: font height

See also: fheight

---

lines — *window height in characters*

The graphics context attribute lines specifies the number of text lines available using the current font.

Initial value: 12

See also: columns and height

---

linestyle — *line style*

The graphics context attribute linestyle specifies the form of drawn lines.

Values: "solid", "dashed", "striped"

Initial value: "solid"

See also: fillstyle and linewidth

---

linewidth — *line width*

The graphics context attribute linewidth specifies the width of drawn lines.

Initial value: 1

See also: linestyle

---

pattern — *filling pattern specification*

The graphics context attribute pattern specifies the particular pattern to be used for drawing when the fillstyle attribute is set to "textured" or "masked".

Values: "black", "verydark", "darkgray", "gray", "lightgray", "verylight", "white", "vertical", "diagonal", "horizontal", "grid", "trellis", "checkers", "grains", "scales", "waves", "*width,#hexdigits*"

Initial value: "black"

See also: fillstyle and Pattern()

pointer — *shape of mouse indicator*

The canvas attribute pointer specifies the shape of the figure that represents the mouse position.

Values: system dependent

Initial value: system dependent

See also: pointercol, pointerrow, pointerx, and pointery

---

pointercol — *mouse location column*

The canvas attribute pointercol gives the horizontal position of the mouse in terms of text columns.

See also: pointer, pointerrow, pointerx, and pointery

---

pointerrow — *mouse location row*

The canvas attribute pointerrow gives the vertical position of the mouse in terms of text lines.

See also: pointer, pointercol, pointerx, and pointery

---

pointerx — *mouse location x-coordinate*

The canvas attribute pointerx specifies the horizontal position of the mouse in pixels.

See also: pointer, pointercol, pointerrow, and pointery

---

pointery — *mouse location y-coordinate*

The canvas attribute pointery specifies the vertical position of the mouse in pixels.

See also: pointer, pointercol, pointerrow, and pointerx

---

pos — *position of window on display screen*

The canvas attribute pos specifies the window position as a string containing comma-separated x- and y-coordinates. Attempts to read or write the position fail if the canvas is hidden.

See also: posx and posy

---

posx — *x-coordinate of window position*

The canvas attribute posx specifies the horizontal window position. Attempts to read or write the position fail if the canvas is hidden.

See also: pos and posy

posy — *y-coordinate of window position*

The canvas attribute posy specifies the vertical window position. Attempts to read or write the position fail if the canvas is hidden.

See also: pos and posx

---

resize — *user resizing flag*

The canvas attribute resize specifies whether the user is allowed to resize the window by interaction with the graphics system.

Values: "on", "off"

Initial value: "off"

---

reverse — *color reversal flag*

The graphics context attribute reverse interchanges the foreground and background colors when it is changed from "off" to "on" or from "on" to "off".

Values: "on", "off"

Initial value: "off"

See also: bg, fg, and drawop

---

row — *text cursor row*

The canvas attribute row specifies the vertical position of the text cursor, measured in characters.

See also: col, cursor, x, and y

---

size — *window size in pixels*

The canvas attribute size specifies the window size as a string containing comma-separated width and height values.

Initial value: enough for 12 lines of 80-column text

See also: columns, height, lines, and width

---

width — *window width in pixels*

The canvas attribute width specifies the width of the window.

Initial value: enough for 80 columns of text

See also: columns, height, and size

---

x — *text cursor x-coordinate*

The canvas attribute x specifies the horizontal position of the text cursor, measured in pixels.

See also: col, cursor, row, and y

---

y — *text cursor y-coordinate*

The canvas attribute y specifies the vertical position of the text cursor, measured in pixels.

See also: col, cursor, row, and x

Appendix D

Keyboard Symbols

Pressing a key on the keyboard produces an Icon event unless the key is a modifier key such as the shift key. Releasing a key does not produce an event.

A key that represents a member of the character set, including traditional actions such as return and backspace, produces a string containing a single character. The control and shift modifiers can affect the particular character produced: For example, pressing control-H produces "\b" (the backspace character).

Other keys, such as function and arrow keys, produce integer-valued events. These values may be referenced symbolically by including the definitions contained in the library file keysyms.icn. The following table lists the values of some of the most commonly used keys.

defined symbol	value	key
Key_PrSc	65377	print screen
Key_ScrollLock	65300	scroll lock
Key_Pause	65299	pause
Key_Insert	65379	insert
Key_Home	65360	home
Key_PgUp	65365	page up
Key_End	65367	end
Key_PgDn	65366	page down
Key_Left	65361	arrow left
Key_Up	65362	arrow up
Key_Right	65363	arrow right
Key_Down	65364	arrow down
Key_F1	65470	function key F1
Key_F2	65471	function key F2
Key_F3	65472	function key F3
Key_F4	65473	function key F4
Key_F5	65474	function key F5
Key_F6	65475	function key F6
Key_F7	65476	function key F7
Key_F8	65477	function key F8
Key_F9	65478	function key F9
Key_F10	65479	function key F10
Key_F11	65480	function key F11
Key_F12	65481	function key F12

Appendix E

Event Queues

Each window has an event queue, which is an ordinary Icon list. Pending() produces an event queue. An event is represented by three consecutive values on the list. The first value is the event code: a string for a keypress event or an integer for any other event. The next two values are Icon integers whose lower-order 31 bits are interpreted as fields having this format:

```
000  0000 0000  0SMC XXXX XXXX XXXX XXXX   (second value)
EEE MMMM MMMM  MMMM YYYY YYYY YYYY YYYY   (third value)
```

The fields have these meanings:

X…X	&x: 16-bit signed x-coordinate value
Y…Y	&y: 16-bit signed y-coordinate value
SMC	&shift, &meta, and &control flags
E…M	&interval, interpreted as $M \times 16^E$ milliseconds
0	unused; should be zero

Coordinate values do not reflect any translation specified by dx and dy attributes; the translation is applied by Event() when an event is read.

A malformed event queue error is reported if an error is detected when trying to read the event queue. Possible causes for the error include an event queue containing fewer than three values, or second or third entries that are not integer values or that are out of range. Only artificially constructed events can produce such errors.

Appendix F

Palettes

Palettes are predefined sets of colors that are used with DrawImage(). Palettes also can be used to limit the colors used by ReadImage().

This appendix documents the contents of Icon's palettes and serves as a reference for the programmer. It is hard, though, to understand a palette just by reading about it. The program palette, which displays and labels the colors of the palette, provides a clearer introduction.

Grayscale Palettes

The g16 palette, shown below, provides 16 shades of gray specified by the hexadecimal characters 0123456789ABCDEF, with 0 being black and F being white in accordance with the usual conventions of image processing.

Actually, g16 is just one member of a family of palettes named g2 through g64, each providing the corresponding number of equally spaced shades from black to white. Each g*n* palette uses the first *n* characters of this list:

0123456789ABC…XYZabc…xyz{}

In every case 0 is black and the rightmost character is white.

For palettes with more than 64 entries, it becomes more difficult and eventually impossible to stay within the printable characters. The remaining grayscale palettes, g65 through g256, use the first *n* characters of &cset instead of the list above.

The c1 Palette

The palette `c1` is designed for constructing color images by hand. It is defined using the Icon color-naming system by the table below.

hue	deep	dark	medium	light	pale	weak
black			0			
gray	1	2	3	4	5	
white			6			
brown	!	p	?	C	9	
red	n	N	A	a	#	@
orange	o	O	B	b	$	%
red-yellow	p	P	C	c	&	|
yellow	q	Q	D	d	,	.
yellow-green	r	R	E	e	;	:
green	s	S	F	f	+	–
cyan-green	t	T	G	g	*	/
cyan	u	U	H	h	`	'
blue-cyan	v	V	I	i	<	>
blue	w	W	J	j	(	)
purple	x	X	K	k	[	]
magenta	y	Y	L	l	{	}
magenta-red	z	Z	M	m	^	=
pink			7			
violet			8			

Note that in the Icon color naming system, “dark brown” and “light brown” are the same as two shades of red-yellow.

Uniform Color Palettes

Programs that compute images can more easily use a palette having RGB colors that are in some sense “equally spaced”. The `c3`, `c4`, `c5`, and `c6` palettes are designed with this in mind, and the `c2` palette can also be considered part of this family. The larger palettes allow better color selection and subtler shadings but use up more of the limited number of simultaneous colors.

For any of these `c`*n* palettes, the palette provides *n* levels of each RGB primary color; letting $m = n - 1$, these levels range from 0 (off) to *m* (full on). The palette also provides all the colors that can be obtained by mixing different levels of the primaries in any combination. Mixing equal levels produces black (0,0,0), white (*m*,*m*,*m*), or a shade of gray. Mixing unequal levels produces colors.

Each `c`*n* palette also provides $(n-1)^2$ additional shades of gray to allow better rendering of monochrome images. $n - 1$ intermediate shades are added in each interval created by the original *n* regular achromatic entries, giving a total of $n^2 - n + 1$ grayscale entries.

The lists below specify the characters used by each palette. The n^3 regular entries are ordered from (0,0,0) to (*m*,*m*,*m*), black to white, with the blue component varying most rapidly and the red component varying most slowly. These are followed in the right column by the additional shades

of gray from darkest to lightest.

c2:	kbgcrmyw	x
c3:	@ABCDEFGHIJKLMNOPQRSTUVWXYZ	abcd
c4:	0123456789ABC…XYZabc…wxyz{}	$%&*–/?@
c5:	\000\001 … yz{|	}~\d\200\201 … \214
c6:	\000\001 … \327	\330\331 … \360

For example, the regular portion of the c3 palette is interpreted this way:

char.	r	g	b	char.	r	g	b	char.	r	g	b
@	0	0	0	I	1	0	0	R	2	0	0
A	0	0	1	J	1	0	1	S	2	0	1
B	0	0	2	K	1	0	2	T	2	0	2
C	0	1	0	L	1	1	0	U	2	1	0
D	0	1	1	M	1	1	1	V	2	1	1
E	0	1	2	N	1	1	2	W	2	1	2
F	0	2	0	O	1	2	0	X	2	2	0
G	0	2	1	P	1	2	1	Y	2	2	1
H	0	2	2	Q	1	2	2	Z	2	2	2

The complete set of grayscale entries in c3, merging regular and extra entries, is @abMcdZ (from black to white).

The sizes of c5 and c6 require that they include some nonprinting characters, so they are better suited for computed images than direct specification.

Appendix G

Changes Since Version 9.0

There are two changes to the graphics facilities ofVersion 9.3 of Icon since the last previously released version, 9.1:

- color names are case-insensitive
- the default for i in WDelay(W, i) is 1.

Version 9.1 of Icon differs from Version 9.0 in several minor ways. This appendix lists the changes that are most significant. Most programs will be unaffected by the changes.

Windows

Windows can no longer be resized by the user unless the new resize attribute is set to "on". Resizing by the program is unaffected.

All windows now handle exposure events when one is blocked awaiting input. Moving a dialog box, for example, no longer leaves a blank area in the program window underneath.

Closing a window now invalidates future access to its canvas by any other bindings that may still exist.

The image of a window produced by image() and tracing now includes information about the window. The form is

window_*i*, *j*(*label)*

where *i* is the serial number of the canvas, *j* is the serial number of the graphic context, and *label* is the label.

Attributes

WAttrib(*name=value*) can no longer produce an error. Unrecognized names or illegal values just cause WAttrib() to fail.

WAttrib(*name*) now always returns a legal value for the names "drawop", "pattern", and "pointer". The default pointer for the X Window S ystem is "left ptr".

Measurements

The default starting point for rectangular areas is now the upper-left pixel of the window, regardless of the setting of the dx or dy attributes. Previously, it was (0,0). This change affects only those programs that set dx or dy.

Ellipses drawn by DrawArc() are one pixel wider and taller than before. This makes the interpretation of width and height by DrawArc() consistent with FillArc() and other procedures.

The default line width is now 1. It previously was 0, giving similar but less predictable results under the X window system.

The appearance of dashed and striped lines has been improved for lines having a width greater than 1.

Text

The text cursor, when enabled, is always drawn in the foreground color. It is a little larger (more prominent) than before.

DrawString() draws all characters exactly as given, with no special interpretation. Previously, the newline character was interpreted specially and the corresponding character of a font could not be drawn. WWrite() and WWrites() still interpret newline, tab, and other formatting characters.

Images

Transparency is now handled properly when reading GIF images.

For readability, spaces may now appear in the front portion of a DrawImage() string.

Dialog Procedures

Dialog procedures now produce a beveled, three-dimensional appearance. Dialog() has been renamed TextDialog(), and some new dialog procedures have been added. ColorDialog() allows the selection of a color. SelectDialog() offers a choice of the items in a list. ToggleDialog() displays a set of independently selectable options.

Appendix H

Changes from Version 8.10

Version 9.0 of Icon introduced major changes with respect to the previous Version 8.10. These changes improve the programming interface, reduce inconsistencies, add utility, and reduce the orientation towards a single underlying platform. Some changes, including function renaming, introduce incompatibilities. This is a one-time effort; no further major changes are planned.

For some changes, especially those affecting attribute names and values, both old and new forms function correctly. The general documentation notes only the new forms; this appendix notes cases where old forms are still implemented for compatibility. New programs, of course, should use the new methods. A further discussion of compatibility and conversion appears at the end of this appendix.

This appendix focuses on major changes visible to the programmer. New features are mentioned only briefly; see the rest of the document for more details. A large number of minor changes are not listed here. These include bug fixes, performance enhancements, and output quality improvements.

Terminology

Graphics facilities no longer are considered a special addition known as X-Icon but rather a standard part of Icon identified by the term *graphics* when a distinction is needed.

The word *window* now has fewer meanings, its main uses now being a data type in Icon or an area on a display monitor. The term *window system* has been replaced by *graphics system* in the Icon documentation. The two types of window attributes are now *canvas* attributes and *graphics context* attributes.

The Library

The Icon program library has been expanded greatly and now includes a large number of graphics procedures. Some of these procedures, such as the Dialog procedures, are documented as part of Icon's graphics repertoire. The distinction between built-in functions and library procedures is less important than before and most Icon graphics programs include link graphics, at least, to access the library. This implies, in turn, that the library must be built and available, and that the IPATH and LPATH environment variables must be set at translation time.

Function Names

All of the graphics functions have been renamed. In most cases, the renaming consists of removing the initial X. XAttrib(), XDefault(), XFlush(), and XSync() were renamed by replacing the X with a W (for *window*).

XBind() has been deleted; its capabilities are subsumed by Clone() and Couple() and by the "canvas=hidden" attribute for creating invisible windows. XUnbind() has been replaced by Uncouple().

Three other functions also were deleted. XClearArea() can be replaced directly by EraseArea() in most applications. The function of XWindowLabel() can be accomplished by setting the label attribute. The pointerx and pointery attributes provide the same information as the former XQueryPointer().

New functions have been added. DrawCircle() and FillCircle() draw or fill a circle or circular arc given the center and radius. DrawPolygon() has been added for symmetry with FillPolygon(). Alert() emits a beep or other signal to attract attention.

Windows as Files

New procedures WOpen(), WClose(), WRead(), WReads(), WWrite(), and WWrites() are similar to open(), close(), and so on. The W procedures, however, deal with graphics windows and use the subject window, &window, by default.

To present a more uniform interface, the documentation now uses these procedures for opening and using windows as files. However, window creation via open(name, "x") and other such direct file accesses are still supported.

A minor change in Version 9 is that open(name, "x") now fails if graphics operations are not configured. Previously, a run-time error occurred.

Measurement Changes

Uniform rules apply to the defaulting of x, y, w, and h for functions whose arguments specify rectangular areas: x and y default to the window's upper-left pixel, while w and h default to the value needed to reach the edge of the window. These defaults are changes in some cases.

Negative values of w and h now are allowed when specifying rectangles, with the effect of extending the rectangle left and/or upward from the starting point.

Angles are now measured in radians instead of 64ths of a degree and are measured in a clockwise instead of counterclockwise direction. This change, which affects DrawArc() and FillArc(), makes the graphics functions consistent with the trigonometric functions.

WAttrib() Changes

WAttrib() now fails if an attempt is made to assign an illegal value, or if an invalid attribute name is given. WOpen() and Clone() behave similarly.

WAttrib() now returns integer values (instead of string values) for numeric attributes such as width.

When clipping is disabled, WAttrib() now returns the null value for the clipping attributes clipx, clipy, clipw, and cliph.

Attribute Changes

The windowlabel attribute has been renamed label. The rows attribute has been renamed lines. The geometry attribute has been removed from the documentation in favor of the pos attribute and the new size attribute. For compatibility, rows, windowlabel, and geometry are still accepted.

Several attribute values have been renamed:

old	new
linestyle=onoff linestyle=doubledash	linestyle=dashed linestyle=striped
fillstyle=stippled fillstyle=opaquestippled	fillstyle=masked fillstyle=textured
iconic=window iconic=icon	canvas=normal canvas=iconic

Again, the old names are still accepted. The iconic attribute has been removed from the documentation, and the previous function of iconic=root is no longer available.

The visual attribute, an X-Windows feature of dubious value, has been removed from the documentation. The more portable depth attribute remains available.

Color Specifications

Decimal and hexadecimal color components are now interpreted in a linear color space. This makes their values device-independent within the limits of Icon's ability to compensate for variations in graphics systems. A new gamma attribute controls the amount of this compensation.

In the color naming system, the adjective pale now means very light instead of light moderate.

ColorValue() can now be used to translate a color even if no window is open; however, system-dependent names may not be recognized in this case.

Font Specifications

There is a new, platform-independent font naming system. System-dependent font names are still allowed, but in the case of ambiguity the Icon interpretation takes precedence.

Pattern Specifications

A set of sixteen predefined pattern names has been added. Decimal pattern specifications have been removed from the documentation; hexadecimal values are preferred. With hexadecimal values, the width of a pattern is no longer limited to 32 pixels.

Images

A new DrawImage() function allows the construction and display of bit-mapped images without the use of external files.

ReadImage() and WriteImage() now use GIF as the default file format. ReadImage() accepts a new, additional argument to quantize an incoming image to a particular color palette. GIF images can also be used with the image and iconimage attributes.

Pixel() now produces string values (comma-separated integer r,g,b values) instead of packed integers. Pixel generation now produces row elements (instead of column elements) consecutively. The background color is now produced for pixel addresses outside the bounds of the window; before, those addresses were omitted.

CopyArea() now allows the omission of window arguments for copying within the subject window. If the source rectangle for CopyArea() extends beyond the window boundary, the outside area is treated as if filled with the background color.

Events and Synchronization

Event() now adjusts the mouse location by the values of the dx and dy attributes to make the keywords &x and &y properly reflect the location's effective address within the window.

The documentation no longer mentions the ability of Pending() to add values to the event queue. The preferred method is to use put(Pending(), a, x, y).

WSync() no longer has a second "discard" argument, which was nonportable and unreliable.

When a graphics program blocks awaiting input from a non-graphics terminal, such as standard input, buffered graphics output is first written to the screen. This eliminates the need for an explicit WFlush() call in such cases.

Keyboard Encodings

The library file xkeysyms.icn, which defines the event codes for special keys such as the Home key, has been renamed to keysyms.icn. Some incorrect values in this file have been fixed, and some symbolic names have changed, but the actual codes returned by Icon have not changed.

Users of the X Window System should read the section about keyboard event codes in Appendix I.

Conversion Assistance

The Icon program library contains files to aid in the transition to Version 9 . Many existing graphics programs can be run under Version 9 by adding the following two lines at the beginning:

```
link xcompat
$include "xnames.icn"
```

The environment variables IPATH and LPATH must be set correctly to access the library.

The xcompat package contains procedures XBind(), XUnbind(), XWindowLabel(), XDrawArc(), and XFillArc() that are compatible with Version 8.10. The file xnames.icn contains preprocessor directives that redefine old names as new names for cases not covered by xcompat.icn.

Ultimately, it is best to revise the source code permanently. For Unix systems, an *ed* script is provided to convert the names of graphics functions called from an Icon program. This script is in the Icon program library as gdata/xnames.ed.

Appendix I

The X Window System

This appendix discusses issues that are specific to implementations of Icon running under the X Window System.

Under X, an Icon program is a *client* that performs graphical I/O on a *server*. The client and server can be the same machine, as when a program runs and displays locally on a workstation, or on different machines. A remote server can be specified by using the display attribute when opening a window.

There are many implementations of X, and different systems provide different features, so this appendix can't say precisely how things will work in all situations.

Color Specifications

Color specifications that are not recognized by Icon are passed to X for interpretation. X servers typically offer large sets of color names, including unusual ones such as orchid and papayawhip.

Color correction is controlled by the gamma attribute. The default value of gamma is based on the color returned by X for the device-independent X color specification RGBi:.5/.5/.5. On older X systems that do not recognize this specification, a configuration default value is used.

The interpretation of RGBi:.5/.5/.5 depends on *properties* associated with the root window. These properties are set by the *xcmsdb* utility. The library program xgamma can be used to set the properties to approximate a particular gamma value.

Font Specifications

In interpreting a font specification, Icon recognizes the following font characteristics and tries to match them as well as possible against the available X fonts:

condensed, narrow, normal, wide, extended
light, medium, demi, bold, demibold
roman, italic, oblique
mono, proportional
sans, serif

The same specification can produce fonts of different appearance on different servers.

If a font specification is not understood or matched by Icon's font naming system, it is passed verbatim to X as a font name. This allows the use of native X font specifications, including wild cards. As a special case, a font specification of "fixed" (without any size or other characteristics) is passed to X as a font name without interpretation.

Drawing Operations

In addition to the special reverse value, the drawop attribute can be set to any of the sixteen standard X *logical functions*. These are listed in the following table. The value of each destination pixel is set by combining the source value (foreground pixel code) with the existing destination pixel code using the bitwise operations indicated below:

drawop=	result
and	source AND dest
andInverted	COMPL(source) AND dest
andReverse	source AND COMPL(dest)
clear	all bits 0
copy	source
copyInverted	COMPL(source)
equiv	COMPL(source) XOR dest
invert	COMPL(dest)
nand	COMPL(source) OR COMPL(dest)
noop	dest
nor	COMPL(source) AND COMPL(dest)
or	source OR dest
orInverted	COMPL(source) OR dest
orReverse	source OR COMPL(dest)
set	all bits 1
xor	source XOR dest

Note that the foreground pixel value cannot be set to an arbitrary bit pattern, but only to the code of an assigned color. The one exception is drawop=reverse, where the XOR of the foreground and background values is used with an XOR logical function.

Images

In ReadImage(), if an image file is not a valid GIF file, an attempt is made to read it as an X Bitmap or X Pixmap file.

In WriteImage(), if the file name ends in .xbm or .XBM, an X Bitmap file written. If the file name ends in .xpm or .XPM, an X Pixmap file is written. If the file name ends in .xpm.Z, a compressed X Pixmap file is written. In all other cases a GIF image is written.

X Resources

Default() returns values registered with the X Resource Manager. These values often are set by an .Xresources or .Xdefaults file.

Keyboard Event Codes

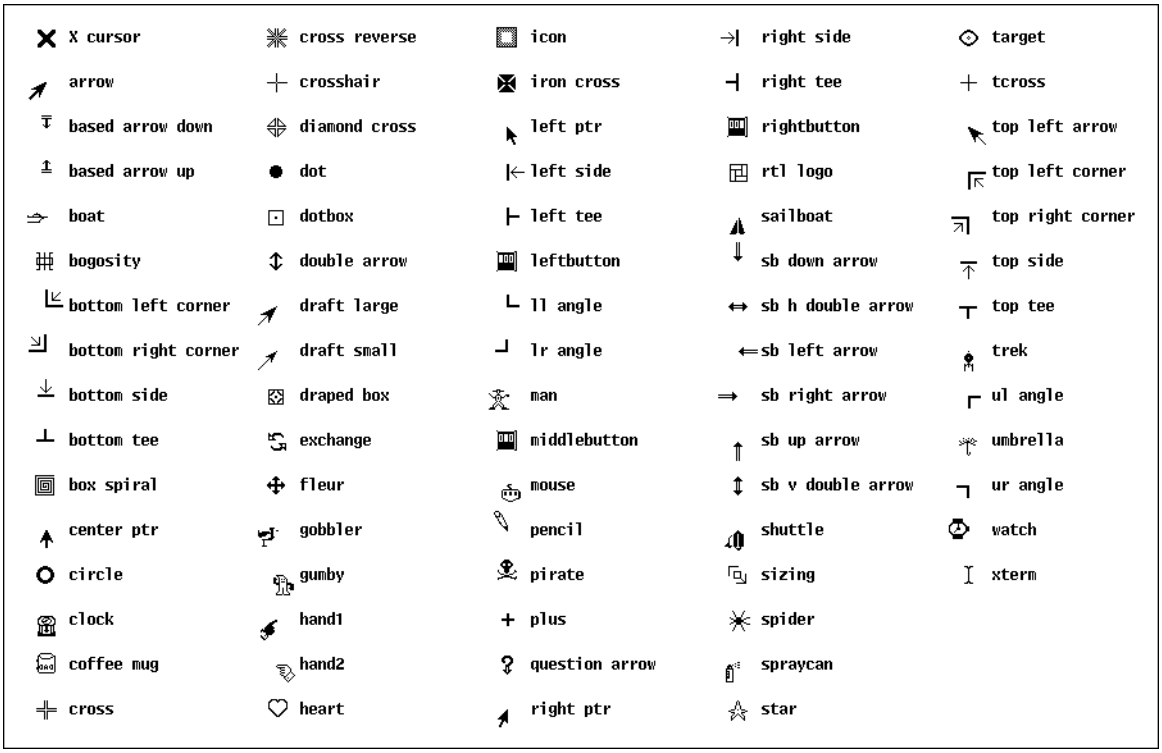
Icon uses X *keysym* codes as event codes. The actual code returned for a particular key depends on the configuration of the X server; this can be altered dynamically by the *xmodmap* utility. For example, the Sun keypad has one key labeled "3", "PgDn", and "R15". Whether this key produces an Icon event "3", Key_PgDn, Key_R15, or even something else depends on the X configuration.

Appendix D lists codes for keys that are found on many keyboards. A larger set is listed in the library file keysyms.icn.

Cursors and Pointers

The text cursor is an underscore character. It is visible only when the cursor attribute is on and the program is awaiting input in WRead() or WReads(). The cursor does not blink and may be hard to locate in a window containing a large amount of text.

The mouse location indicator, set by the pointer attribute, is selected from the X cursor font. The following values are accepted:



The default mouse location indicator is "left ptr".