Icon Graphics—Introduction

Facilities for graphical programming in Icon evolved in the period 1990-1994.

A philosophy of Icon is to insulate the programmer from details and place the burden on the language implementation. The graphics facilities were designed with same philosophy.

Icon's graphical facilities are built on the X Window System on UNIX machines. On Microsoft Windows platforms the facilities on built on the Windows API.
Window basics

Before any graphical operations can be done, a window must be opened.

Here is a complete program that opens a window with a specific width, height, and label:

```plaintext
link graphics
procedure main() # win1
    WOpen("height=100","width=300",
           "label=A Window")
    WDone()
end
```

As a rule, graphics programs should link graphics.

On UNIX the program can be compiled with icont, as usual. Use wicont on Windows.

On a Windows platform, here's the result:

![Window Result]

WOpen() accepts zero or more window attributes as arguments. Attributes may be specified in any order.

WDone() waits until a q or Q is typed in the window.
Window basics, continued

Window attributes can be queried with \texttt{WAttrib(s1, s2, \ldots)}. The value of each named attribute is generated.

\texttt{WWrite()} is like \texttt{write()}, but sends output to the window.

Example:

\begin{verbatim}
link graphics
procedure main() # win2
    WOpen("height=100","width=300",
           "label=A Window")
    every WWrite(WAttrib("height", "width",
                      "size", "label"))
    WDone()
end
\end{verbatim}

Resulting window:

In essence, \texttt{WWrite()} treats the window as a scrolling text window.

\texttt{write()} could be used instead of \texttt{WWrite()}; output would then go to the "console".
Coordinate system

The coordinate system is integer based with (0,0) in the upper left corner of the window. Here are the corner points for a window with size=200, 100:
Drawing points

The simplest drawing primitive is \texttt{DrawPoint(x, y)}, which draws one pixel at the specified coordinates in the foreground color (black, by default).

Example:

\begin{verbatim}
link graphics
procedure main() # dp1
    WOpen("size=300,100")
    every x := 0 to 299 by 3 do
        DrawPoint(x, 50)   # horizontal
    every y := 0 to 99 by 7 do
        DrawPoint(150, y)  # vertical

    WDone()
end
\end{verbatim}

Result:
Drawing points, continued

Some fun with randomly drawn points:

link graphics

$define Height 100       # symbolic constants
$define Width 300        # via preprocessor

procedure main() # dp2
    WOpen("size=", Width, ",", Height)
    repeat {
        DrawPoint(?Width-1, ?Height-1)
    }
    WDone()
end

Another angle:

link graphics
$define Height 100
$define Width 300

procedure main(args) # dp3
    WOpen("size=", Width, ",", Height)
    N := args[1] | 1
    repeat {
        x := y := 0
        every 1 to N do x += ?(Width/N)
        every 1 to N do y += ?(Height/N)
        DrawPoint(x, y)
    }
    WDone()
end
Drawing lines

`DrawLine(x1, y1, x2, y2)` draws a line between the points \((x1, y1)\) and \((x2, y2)\), inclusive.

```link graphics
procedure main(args) # dl2
  WOpen("size=300,100")
  WAttrib("linewidth=" || args[1])
  DrawLine(10, 10, 290, 90)
  DrawLine(10, 90, 290, 10)
  DrawLine(150, 10, 150, 90)
  WDone()
end
```

When run with no arguments, a default linewidth of 1 is used:

Here is a linewidth of 5:
Drawing lines, continued

An arbitrary number of coordinate pairs can be passed to DrawLine. It draws a line between the first and second points, then the second and third points, etc.

procedure main() # dl3
    WOpen("size=300,100")
    DrawLine(100,10,200,10,100,90,200,90)
    WDone()
end

Result:

![Image of a drawn figure]

Icon's *list invocation* syntax is often used with drawing functions that accept a variable number of arguments:

procedure main() # dl3a
    WOpen("size=300,100")
    zpts := [100,10,200,10,100,90,200,90]
    DrawLine!zpts # "list invocation"
    WDone()
end

A related function is DrawSegment, which draws disjoint segments for each pair of coordinate pairs.
Drawing lines, continued

Problem: Write a program that produces an approximation of this image:
Drawing rectangles

The function DrawRectangle(x, y, w, h) draws the outline of a rectangle.

With a line width of 1, the upper left corner is at (x, y) and the lower right corner is at (x+w, y+h).

```
procedure main(args) # dr1
  WOpen("size=300,150")
  x := y := 0
  every h := 10 to 35 by 5 do {
    DrawRectangle(x, y, h*2, h)
    x := h*2
    y := h
  }
  WDone()
end
```

Result:
Drawing rectangles, continued

FillRectangle(x, y, w, h) is just like DrawRectangle but it produces a rectangle filled with the foreground color.

A related function is EraseArea, which accepts the same arguments and fills the rectangular area with the background color (white, by default).

```pascal
procedure main(args) # dr2
  WOpen("size=300,150")
  x := y := 0
  every h := 10 to 35 by 5 do {
    FillRectangle(x, y, h*2, h)
    EraseArea(x, y, 5, 5)
    x +:= h*2
    y +:= h
  }
  WDone()
end
```

Result:
Drawing rectangles, continued

Appendix I in the text covers some painful but important details about the rendering of various figures.

One example of "interesting" behavior is the difference in the rectangular area when drawn with DrawRectangle versus FillRectangle:

<table>
<thead>
<tr>
<th></th>
<th>1x1</th>
<th>2x2</th>
<th>3x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawRectangle</td>
<td>■</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>FillRectangle</td>
<td></td>
<td>■</td>
<td></td>
</tr>
</tbody>
</table>
Drawing circles

Circles are drawn with \texttt{DrawCircle(x, y, radius)}:

```plaintext
procedure main(args) # dcl
   WOpen("size=200,200")

   width := 1
   every r := 3 to 100 by 20 do {
      DrawCircle(100, 100, r)
      WAttrib("linewidth=" || (width +:+= 2))
   }

   WDone()
end
```

Result:
Drawing circles, continued

The previous example used some defaults. \texttt{DrawCircle} is actually more general:

\[ \text{DrawCircle}(x, y, r, \text{start}, \text{extent}) \]

This draws a circular arc centered at \((x, y)\) with radius \(r\) starting at \text{start} radians and continuing through \text{extent} radians. (Recall that \(2\pi\) radians equals 360 degrees.)

\text{start} is measured with zero at 3 o'clock. Positive values for \text{start} and \text{extent} indicate a clockwise direction; negative values indicate counter-clockwise direction.

\begin{align*}
\text{DrawCircle}(..., 0, \pi) & \quad \text{DrawCircle}(..., \pi, \pi) \\
\text{DrawCircle}(..., \pi/3, \pi*0.9) & \quad \text{DrawCircle}(..., -\pi/8, -\pi*0.5) \\
\end{align*}
Drawing circles, continued

Here is a simple-minded test program that exercises DrawCircle and its counterpart, FillCircle:

```plaintext
procedure main(args) # dc3
    WOpen("size=200,240",
        "linewidth=10")

    WWrite(repl("\n",30))
    repeat {
        EraseArea()

        WWrite("f/d, start, extent? ")

        args := split(WRead())
        p := if get(args) == "f"
            then FillCircle else DrawCircle

        every !args *:= (2*&pi)/360

        p!([100,100,90]|||args)
        WRead()
    }
end

Notes:
    Via defaults, EraseArea() erases the entire window.

    WRead() reads a line of input typed directly into the window.
```