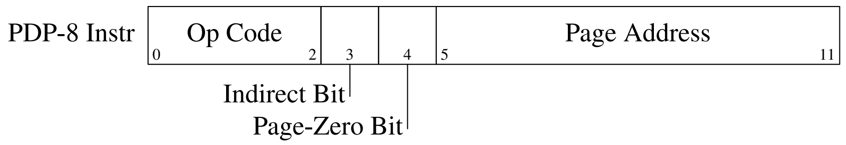
DFORMAT — A Program for Typesetting Data Formats

Jon L. Bentley

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

ABSTRACT

Data formats ranging from computer words to packets on a data network are often described by pictures composed of rectangles. The PDP-8, for instance, uses this instruction format:



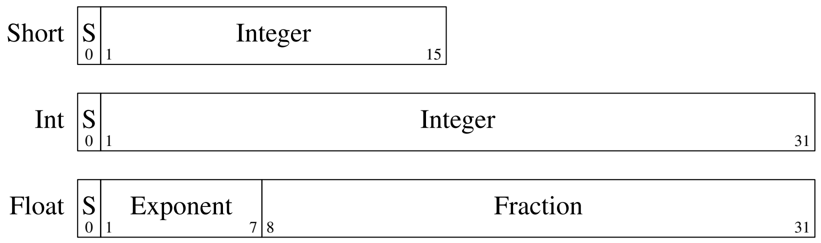
The DFORMAT program allows such diagrams to be included in TROFF documents. The above diagram is described as

```
.begin dformat
style bitwid .3
PDP-8 Instr
        0-2 Op Code
        3 Indirect Bit
        4 Page-Zero Bit
        5-11 Page Address
.end
```

DFORMAT is implemented as a preprocessor for the PIC language. Its implementation (about 100 lines of AWK) is included in this paper.

1. Simple Pictures

This picture shows the format that the IBM System/360 uses for storing short integers, integers, and floating-point numbers (or shorts, ints, and floats, in C terminology):



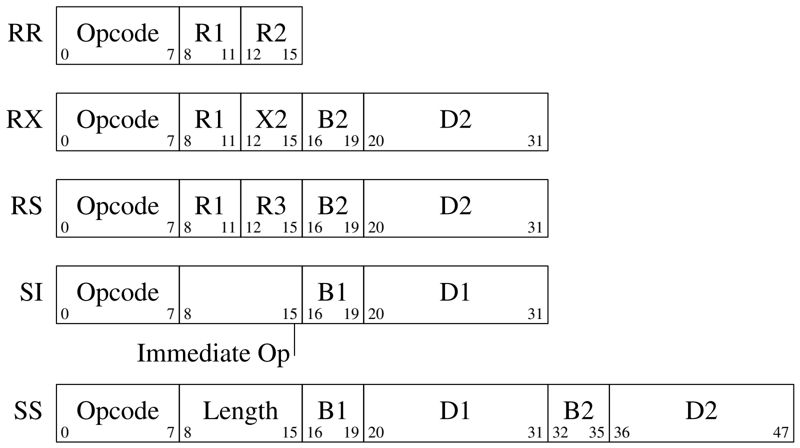
Each line describes a *record* that contains several *fields*. The figure was described by this text in the input file:

```
.begin dformat
style bitwid 0.12
Short
        0 S
        1-15 Integer
Int
        0 S
        1-31 Integer
Float
        0 S
        1-7 Exponent
        8-31 Fraction
.end
```

The `.begin dformat` and `.end` lines delimit the DFORMAT input. The `style` line states that bits are rendered as 0.12 inches wide. Lines that start in column 1 name the records; subsequent lines that begin with white space give the field widths and names, in order. Because DFORMAT is a PIC preprocessor, the document is compiled with a pipeline like

```
dformat paper.in | pic | tbl | eqn | troff -ms >paper.out
```

The System/360 has five instruction formats:

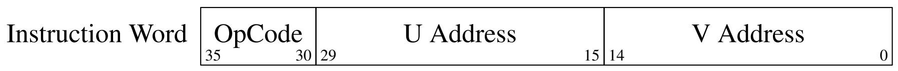


Because the “Immediate Op” text in the SI instruction is too long to fit in its box, DFORMAT places the text below the box and connects it with a line (as it did twice in the PDP-8 instruction format in the abstract). This picture is described as follows:

```
.begin dformat
style bitwid 0.08
RR
        0-7 Opcode
        8-11 R1
        12-15 R2
RX
        0-7 Opcode
        8-11 R1
        12-15 X2
        16-19 B2
        20-31 D2
...
.end
```

The `style` command can be viewed as an assignment of the value 0.08 inches to the parameter `bitwid`. The widest instruction format of 48 bits is therefore 3.84 inches across. The assignments persist between DFORMAT diagrams in a document; in most documents, therefore, the `bitwid` parameter is set only in the first picture.

Bits can be numbered with zero at the right; here is the instruction format of the UNIVAC 1103A:



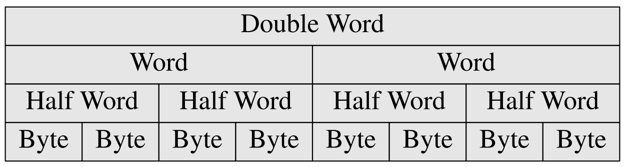
It was produced by this input:

```
.begin dformat
style bitwid 0.1
Instr
        35-30 OpCode
        29-15 U Address
        14-0 V Address
.end
```

These examples illustrate the typical use of DFORMAT. Simple pictures are simply described; this level of detail is sufficient for most users. Section 2 describes additional parameters for controlling pictures, and Section 3 contains several more sophisticated examples. If the built-in parameters aren't enough, Section 4 describes how you might tinker with the implementation.

2. Adjusting Parameters

Many computer systems have memory organized like this:

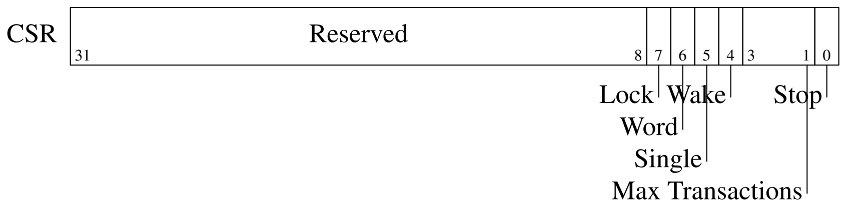


The picture was drawn by this DFORMAT description:

```
.begin dformat
style fill on
style bitwid 0.05
style recspread 0
style addr off
style recht 0.2
noname
        0-63 Double Word
noname
        0-31 Word
        32-63 Word
noname
        0-15 Half Word
        16-31 Half Word
        32-47 Half Word
        48-63 Half Word
 ...
.end
```

Setting the `fill` parameter to `on` produces filled boxes. The `bitwid` parameter of 0.05 makes the 64-bit record 3.2 inches wide. The `recspread` parameter is the spread between records; the value of 0 causes the records to be stacked with no intervening space. Assigning `off` to `addr` turns the addresses off; the default assignment is `both`, but it can also be set to either `left` or `right`. The `recht` parameter ensures that each record is depicted by a rectangle 0.2 inches high. All records have the name `noname`, so no text appears to the left of the records.

Diagrams with many long names in short fields can lead to esthetic problems. This picture, for instance,



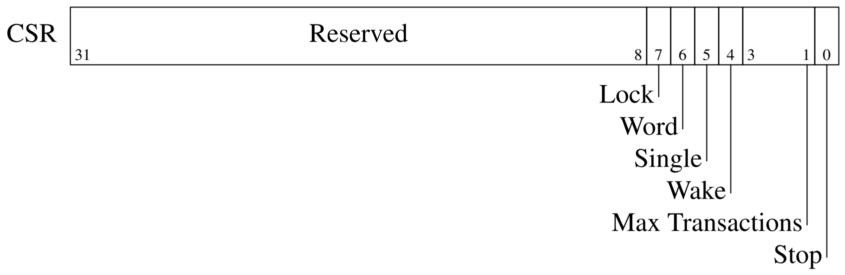
was described by this input:

```
.begin dformat
style bitwid 0.125
CSR
        31-8    Reserved
        7       Lock
        6       Word
        5       Single
        4       Wake
        3-1     Max Transactions
        0       Stop
.end
```

The field names that do not fit in the boxes are placed in channels below the fields, with as many names as possible in each channel. The lines that connect a field description to its box may pass through other text. If you prefer that lines not pass through text, then you can set the variable

```
style linethrutext 0
```

which results in this picture:



This table contains a complete list of available parameters.

NAME	INITIAL VALUE	EXPLANATION
bitwid	0.125	Width of 1 bit in inches
charwid	0.07	Width of 1 character in inches
recht	0.3	Height of boxes, in inches
recspread	0.15	Spread between boxes, in inches
textht	0.167	Height of text lines below box, in inches
linedisp	0.04	Distance of line from right of box, in inches
linethrutext	1	Nonzero if lines may pass through text
addrht	0.055	Height of addresses above box, in inches
addrdelta	4	Delta point size for printing bits ($0 \le d \le 9$)
addr	both	Where to print addresses (left, right, both, off)
fill	off	On to fill boxes, off for unfilled boxes
shaded	off	color spec to fill boxes, off for unfilled boxes

The variables `addrht` and `addrdelta` control the addresses printed at the corners of the field boxes; `addrht` is their height in inches above the bottom of the box and `addrdelta` is the decrease in point size from the default font size (so if it has the value 4 and the point size is 10, the addresses will be printed in 6-point). The variable `charwid` is the width of an average character, which is used to decide whether a text string will fit in its box (0.07 inches is about right for 10-point Times Roman, but any such value is only an approximation; notice the widths of MMMMM and iiiii). If text does overflow, it is placed beneath the boxes in channels `textht` inches high, and vertical lines are drawn down from `linedisp` inches from the right end of the box. If `linethrutext` is zero, then the connecting lines should not pass through other text. If `fill` is `on`, then records are drawn as filled boxes. To adjust the darkness of filled boxes, one passes an assignment through to PIC:

```
pic fillval = 0.9
```

Lower values produce darker backgrounds.

The values of variables are retained from one DFORMAT display to another. This allows you to define a style for a document by setting all values in the first display. To allow you to use a different style in a single picture, the old value of a variable is stored when a new value is assigned. The old value of the variable `bitwid`, for instance, may be restored by an assignment of the form

```
style bitwid reset
```

The address of a field is typically specified by a pair of integers, as in 4-7. The single integer 4 is an abbreviation for the single-bit range 4-4. These two formats account for most fields, but two other kinds of specifications support more exotic fields:

FORMAT	INTERPRETATION
i	Field *i*..*i*, width 1
i-*j*	Field *i*..*j*, width $j - i + 1$
l-*r*-*w*	Field *l*..*r*, width *w*
l-*r*-*w*-*t*	Field *l*..*r*, width *w*, box type *t*

The variables *i* and *j* must be integers, *w* may be a real, and *l* and *r* are strings. Under the third format, the field specified by

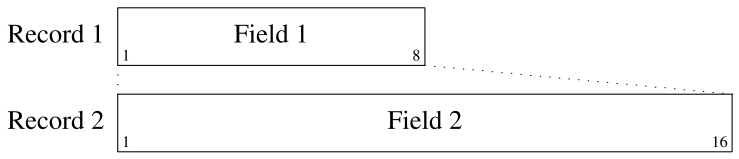
```
lo-hi-3
```

has left index "lo", right index "hi" and width 3 bits. Under the fourth format, each field may be given a type of "dotted" or "dashed" or "solid" (as in PIC, "dot" and "dash" are acceptable abbreviations). These conventions are illustrated in this nonsense figure described by:

```
.begin dformat
style bitwid .3
noname
        1 A
        2-3 B
        lo-hi-3 C
        --3-invis ...
        88-90-3-dash D
.end
```

Spaces are not allowed in the indices, so if you want spaces in the output you will have to resort to vile trof-fery such as using "\|\|" to represent two adjacent half spaces. We'll soon see that elaborate uses of this notation are somewhat clumsy, but they do get they job done.

DFORMAT has two final features that are useful for connecting fields in one record to fields in another record: field names and the ability to pass commands through to PIC. This picture



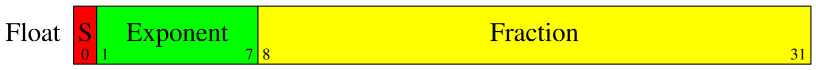
was produced by this description.

```
.begin dformat
style bitwid .2
Record 1
   F1:  1-8 Field 1
Record 2
   F2:  1-16 Field 2
pic line dotted from F1.sw to F2.nw
pic line dotted from F1.se to F2.ne
.end
```

Field lines have leading white space; if the first string in such a line ends in a colon, it is interpreted as a name of the corresponding PIC box (recall that PIC names must begin with capital letters). If the first field on a line is the word ‘‘pic’’, then the rest of the line is passed through to be processed by PIC. We’ll shortly see an application of these mechanisms.

3. Using groff’s color attributes

gPIC and groff have the ability to assign color to objects. DFORMAT uses the `shaded` style parameter to produce colored boxes.

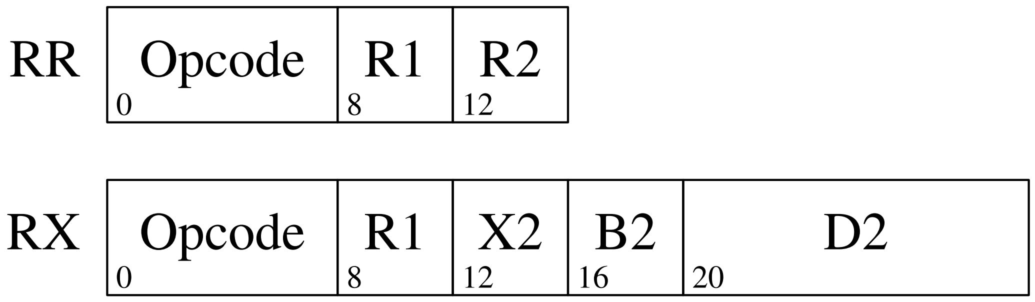


was produced by this description.

```
.begin dformat
style bitwid 0.12
Float
style shaded red
        0 S
style shaded green
        1-7 Exponent
style shaded yellow
        8-31 Fraction
style shaded off
style bitwid reset
.end
```

4. Additional Examples

DFORMAT can be used to prepare overhead transparencies and other material that requires oversize text. This large version of two System/360 instruction formats



was described in this form:

```
.ps +10
.begin dformat
style bitwid 0.15
style recht 0.6
style recspread 0.3
style addrdelta 8
style addrht 0.1
style addr left
RR
        0-7 Opcode
        8-11 R1
        12-15 R2
RX
        0-7 Opcode
        8-11 R1
        12-15 X2
        16-19 B2
        20-31 D2
.end
.ps -10
```

The TROFF `.ps +10` command increments the point size from 10 points to 20 points. The series of `style` commands sets the parameters to be more appropriate for this larger format. (I started by doubling any parameter that ‘‘looked funny’’ in the regular form, and then twiddled them to look a little better.) The assignment of `left` to the parameter `addr` causes only the left bit addresses to be printed (printing both looks a little crowded).

The next figure shows the packets used in a data communications network. It represents the most complex kind of figure that can (well, more accurately, should) be drawn with DFORMAT. The description uses complex field descriptions, named fields, PIC pass-throughs, and embedded EQN.

```
.begin dformat
style bitwid 0.08
style charwid 0
style recspread 0.3
noname
        --16 Frame
        --16 Frame
  A1:   --16 Frame
        --16 Frame
        --8-dash ...
noname
  A2:   8--8 Flags
        8--8 Status
        --8 @roman Chunk sub 1@
  B1:   --8 @roman Chunk sub 2@
        --8-dash ...
        --8 @roman Chunk sub m@
        16--16 CRC
  A3:   8--8 Flags
noname
  B2:   8--8 @roman Data sub 1@
        8--8 @roman Data sub 2@
        8--8 @roman Data sub 3@
        8--8 @roman Data sub 4@
        --8-dash ...
        8--8 @roman Data sub {n-1}@
        8--8 @roman Data sub n@
        6--6 Length
  B3:   10--10 Channel #
pic line dotted from A1.sw to A2.nw
pic line dotted from A1.se to A3.ne
pic line dotted from B1.sw to B2.nw
pic line dotted from B1.se to B3.ne
.end
```

5. DFORMAT Implementation

I hope that many users of DFORMAT will rarely need material beyond that contained in Section 1, and that Sections 2 and 3 cover most of the exceptions. If you want to go even further, there are no additional bells and whistles in DFORMAT; you must modify the program. This section begins by describing a miniature version of the program and then presents the entire source code.

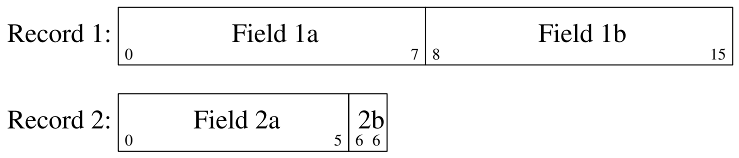
This trivial version of DFORMAT draws simple data formats. It does not support parameters and style assignments, nor does it place long strings below their boxes.

```
awk '
inlang == 0 { if ($0 !~ /^\.begin[ \t]/ || $2 != "dformat") print
              else { inlang = 1; print ".PS"; boxacnt = 0 }
              next
            }
/^\.end/    { inlang = 0; print ".PE"; next }
/^[^ \t]/   { printf "BoxA: box invis ht 0.3 wid 0"
              if (boxacnt++) printf " with .n at BoxA.s - (0,0.15)"
              printf "\n"
              printf " \"%s: \" rjust at BoxA.w\n", $0
              printf " BoxB: box invis ht 0.3 wid 0 at BoxA\n"
              next
            }
/./         { range = $1; $1 = ""
              gsub(/¿[ \t]+/, ""); gsub(/[ \t]+$/, ""); text = $0
              n = split(range, x, "-")
              rlo = x[1]
              rhi = (n >= 2) ? x[2] : rlo
              rwid = rhi - rlo + 1
              printf " BoxB: box %s ht .3 wid %g with .w at BoxB.e\n",
                      btype, rwid*.2
              printf "    \"%s\" at BoxB.c\n", text
              printf "\t\" \\s-4%s\\s+4\" ljust at BoxB.sw + (0,.06)\n", rlo
              printf "\t\"\\s-4%s\\s+4 \" rjust at BoxB.se + (0,.06)\n", rhi
            }
' $*
```

This sample input

```
.begin dformat
Record 1
        0-7 Field 1a
        8-15 Field 1b
Record 2
        0-5 Field 2a
        6 2b
.end
```

produces this picture



by making this PIC file:

```
.PS
BoxA: box invis ht 0.3 wid 0
 "Record 1: " rjust at BoxA.w
 BoxB: box invis ht 0.3 wid 0 at BoxA
 BoxB: box  ht .3 wid 1.6 with .w at BoxB.e
    " Field 1a" at BoxB.c
        " \s-40\s+4" ljust at BoxB.sw + (0,.06)
        "\s-47\s+4 " rjust at BoxB.se + (0,.06)
 BoxB: box  ht .3 wid 1.6 with .w at BoxB.e
    " Field 1b" at BoxB.c
        " \s-48\s+4" ljust at BoxB.sw + (0,.06)
        "\s-415\s+4 " rjust at BoxB.se + (0,.06)
BoxA: box invis ht 0.3 wid 0 with .n at BoxA.s - (0,0.15)
 "Record 2: " rjust at BoxA.w
 BoxB: box invis ht 0.3 wid 0 at BoxA
 BoxB: box  ht .3 wid 1.2 with .w at BoxB.e
    " Field 2a" at BoxB.c
        " \s-40\s+4" ljust at BoxB.sw + (0,.06)
        "\s-45\s+4 " rjust at BoxB.se + (0,.06)
 BoxB: box  ht .3 wid 0.2 with .w at BoxB.e
    " 2b" at BoxB.c
        " \s-46\s+4" ljust at BoxB.sw + (0,.06)
        "\s-46\s+4 " rjust at BoxB.se + (0,.06)
.PE
```

The complete program produces similar PIC output; here is the source code.

```
#! /bin/awk -f
# mailto:jlb@research.bell-labs.com

function error(s)
{
    print "dformat error: " s " near input line " NR | "cat 1>&2"
}

BEGIN {
    s =   "recht 0.3    addrht 0.055    recspread 0.15 "
    s = s "charwid 0.07 textht 0.167    addrdelta 4 "
    s = s "bitwid 0.125 linedisp 0.04   addr both "
    s = s "fill off     linethrutext 1 "
    s = s "shaded off"
    n = split(s, x)
    for (i = 1; i <= n - 1; i += 2) oparm[x[i]] = parm[x[i]] = x[i+1]
}

inlang == 0 {
    if ($0 !~ /^\.begin[ \t]/ || $2 != "dformat") {
        print
    } else {
        inlang = 1
        print ".PS"
        boxacnt = 0
        if (firstpic != 1) {
            firstpic = 1
            print "fillval = 0.1"
        }
    }
    next
}

/^\.end/ {
    inlang = 0
    print ".PE"
    next
}

$1 == "style" {
    if (!($2 in parm)) {
        error("unrecognized name: " $2)
    } else if ($3 == "reset") {
        t = oparm[$2]
        oparm[$2] = parm[$2]
        parm[$2] = t
    } else {
        oparm[$2] = parm[$2]
        parm[$2] = $3
        #error("set shaded to: " $3)
    }
    next
}

$1 == "pic" {
    $1 = ""
    print $0
    next
}

/^[^ \t]/ {
    printf "BoxA: box invis ht %g wid 0", parm["recht"]
    if (boxacnt++) {
        printf " with .n at BoxA.s - (0, %g)",
            parm["recspread"] + maxdy * parm["textht"]
    }
    printf "\n"

    maxdy = sumboxlen = 0
    gsub(/[ \t]+$/, "")
    if ($0 != "noname") {
        printf " \"%s  \" rjust at BoxA.w\n", $0
        printf " box invis with .e at BoxA.w ht 0 wid %g\n",
            parm["charwid"] * (length($0) + 3)
    }
    printf " BoxB: box invis ht %g wid 0 at BoxA\n", parm["recht"]

    next
```

```
}

/./ {
    boxname = ""
    if ($1 ~ /:$/) {
        boxname = substr($1, 1, length($1) - 1)
        $1 = ""
        $0 = " " $0
    }
    range = $1
    $1 = ""
    gsub(/^[ \t]+/, "")
    gsub(/[ \t]+$/, "")
    text = $0

    n = split(range, x, "-")
    rlo = x[1]
    rhi = (n >= 2) ? x[2] : rlo
    cwid = (rhi >= rlo) ? rhi - rlo + 1 : rlo - rhi + 1
    rwid = (n >= 3) ? (0 + x[3]) : cwid
    btype = x[4]
    if (btype !~ /^(dot|dash|invis)/) {
        btype = "solid"
    }

    textlen = parm["charwid"] * length(text)
    boxlen = parm["bitwid"] * rwid

    dy = 0
    if (textlen > boxlen) {
        # set dy, the channel for this text
        chan[maxdy + 1] = -999
        for (dy = 1; chan[dy] + textlen > sumboxlen; dy++);
        if (dy > maxdy) {
            maxdy = dy
        }
        if (parm["linethrutext"] == 0) {
            for (k = 1; k <= dy; k++) {
                chan[k] = sumboxlen+boxlen
            }
        } else {
            chan[dy] = sumboxlen
        }
    }
    sumboxlen += boxlen

    fill = ""
    if (parm["fill"] == "on") {
        fill = " fill "
    }

    shaded = ""
    if (parm["shaded"] != "off") {
        shaded = " shaded "
        shaded = shaded "\"" parm["shaded"] "\" "
    }

    if (boxname != "") {
        printf "  %s:", boxname
    }
    printf "  BoxB: box %s %s %s ht %g wid %g with .w at BoxB.e\n",
        shaded, fill, btype, parm["recht"], boxlen

    if (dy == 0) {
        printf "    \"%s\" at BoxB.c\n", text
    } else {
        if (rwid < 2) {
            start = "BoxB.s"
        } else {
            start = "BoxB.se - (" parm["linedisp"] ",0)"
        }
        printf "    line from %s down %g\n", start, dy * parm["textht"]
        printf "    \"%s\\|\" at last line .s rjust\n", text
        printf "    box invis with .e at last line .s ht 0 wid %g\n",
            textlen
    }

    if (parm["addr"] ~ /^(left|right|both)$/) {
        dp = int(parm["addrdelta"])      # Delta Point size
        if (dp < 0 || dp > 9) {
            error("bad addrdelta value: " dp)
        }

        dah = parm["addrht"]             # Delta Addr Height
        pb = parm["addr"]                # Parameter for Bits

        if (rlo == rhi) {
            printf "    \"\\s-%d%s\\s+%d\" at BoxB.s + (0,%g)\n",
                dp, rlo, dp, dah
        } else {
            if (pb == "left" || pb == "both") {
                printf "\t\"\\|\\s-%d%s\\s+%d\" ljust at BoxB.sw + (0,%g)\n",
                    dp, rlo, dp, dah
            }
            if (pb == "right" || pb == "both") {
                printf "\t\"\\s-%d%s\\s+%d\\|\" rjust at BoxB.se + (0,%g)\n",
                    dp, rhi\, dp, dah
            }
        }
    }
}

END {
    if (inlang) {
        error("eof inside begin/end")
    }
}
```

I built DFORMAT for a colleague who wanted to include data formats in a document. He described the problem to me on a Friday afternoon. I wrote the first version in a couple of hours on Saturday; it was a tad larger and dirtier than the simple version presented above. After my colleague agreed that the output was in the right ballpark, I spent six hours on Sunday adding parameters, error checking, and several other fancinesses. The program had just one user for a couple of months, but I described it to a number of other people. After several requests for the code, I spent a few days polishing the program and writing this document. Thus the paper in your hand now, complete with code, represents roughly one staff-week of programmer time.

Were one to insist on putting more effort into this project, there are several obvious choices. Field names that overflow the boxes are usually handled pretty well by the simple algorithm in the current program, but some users might like to be able to control the process explicitly. One could build a DFORMAT-like language for other typesetting systems, such as TEX. If you don't have access to sophisticated output devices, you could write a program to translate from a DFORMAT-like language into a character array:

```
       ----------------
    RR |Opcode| R1| R2|
       ----------------
       0        8   12

       --------------------------------
    RX |Opcode| R1| X2| B2|      D2     |
       --------------------------------
       0        8   12  16  20
```

Acknowledgments

I am grateful for the helpful comments of Sandy Fraser, Brian Kernighan, and Ravi Sethi.

Summary of Features Added Since Original Version

The `fill` style parameter now permits filled boxes, and the `linethrutext` parameter avoids lines that pass through text.