Advanced Use of Lisp's FORMAT Function

Gene Michael Stover

created 22 January 2004
updated Saturday, 30 October 2004

Contents

1 Introduction

Lisp's FORMAT function is analogous to C's `printf` function, but FORMAT can do a whole lot more. FORMAT can iterate, use conditionals, process nested format strings, & recurse into format strings embedded into its arguments.

The more-or-less official documentation for FORMAT is at the Common Lisp Hyperspec[X3J]. There is a complete if terse description of it in Paul Graham's ANSI Common Lisp[Gra96], but my favorite description of FORMAT is in Franz's Common Lisp The Reference[Inc88].

Knowing the details of how something works isn't the same as knowing how to use it well. Good applications of FORMAT aren't obvious from its documentation. Thus this article.

2 Wanted & To Do

Read the original article about FORMAT[Wat89]. Possibly incorporate ideas from it, or refer to them. Definitely refer to it in this article so people know it exists. (I didn't know it existed at all even though I searched the web for such articles before I wrote this one. I wouldn't have known about the original article if Brian T. Rice hadn't told me.)

Wanted: Clever way to make FORMAT print a GUID with the hypens in the customary locations. You may assume a GUID is stored internally as a very large integer, as a string of characters, or possibly as a vector of bits, whichever is most convenient for you. But the hypens are not stored in the internal representation of the GUID.

To Do: Show the # and V options to format strings.

3 How to Print a Row

Let's say you have a list containing the data for one row in a table, & you want to use FORMAT to print that row. In other words, you have something like this:

```
(1 green ribbit)
```

and you want to print something like this:

```
   number     color     noise
   ------  --------  --------
        1     green    ribbit
```

Notice that the third line, which has "green" & "ribbit" in it, is from the list that we have.

The first column has a width of six; the second & third columns have widths of eight. The first column is prefixed by three spaces. The columns are separated by two spaces.

Here's one way to print that line, assuming the data for the row is in a list called ROW:

```
(format t "~&   ~6D  ~8A  ~8A"
        (first row) (second row) (third row))
```

Here are some examples, using NIL so that FORMAT will return strings.

```
> (setq row (list 1 'green 'ribbit))
(1 GREEN RIBBIT)
```

```
> (format nil "~&   ~6D  ~8@A  ~8@A"
          (first row) (second row) (third row))
"        1     GREEN    RIBBIT"
```

It's a bummer that we have to separate the list into parts before calling FORMAT. We could avoid that by applying APPLY to the situation, but that would require more typing & some *cons*ing. A better (but not very good) way to make FORMAT do that work with the `~?` directive, like this:

```
> (format nil "~?"
          "~&   ~6D  ~8@A  ~8@A"
          row)
"        1     GREEN    RIBBIT"
```

A better way to make FORMAT do the work is with the `~{` directive.

```
> (format nil "~&   ~{~6D  ~8@A  ~8@A~}" row)
"        1     GREEN    RIBBIT"
```

This is the best way I know so far to print a single row if you have the items of a row in a list.

There's one more twist: What if the row list contains an item we don't want to print? We can use the `~*` directive to skip the items we don't want to print.

```
> (format nil "~&   ~{~6D  ~*~8@A  ~8@A~}"
          (list 2 'skip-me 'yellow 'buzz))
"        2    YELLOW      BUZZ"
```

If you want to print an entire table or report, don't stop reading here. Also read How to Print a Table (4).

4 How to Print a Table

This builds on How to Print a Row (3), but now you have all the data for your report in a single list with one element for each row. In other words, maybe you have the data for your report in a list of lists, like this:

```
> (setq table
        (list (list 1 'green 'ribbit)
              (list 21 'yellow 'buzz)
              (list 17 'orange 'meow)
              (list 29 'purple 'silence)))
((1 GREEN RIBBIT) (21 YELLOW BUZZ)
 (17 ORANGE MEOW) (29 PURPLE SILENCE))
```

You want to print a report like this:

```
   number     color     noise
   ------  --------  --------
        1     green    ribbit
       21    yellow      buzz
       17    orange      meow
       29    purple   silence
```

We can use the `~{` directive to apply a format string to each item in the table. It's a small twist on the second-to-last example in Section 3 How to Print a Row. We use the "`:`" option on the `~{` directive so that the format string inside the `~{` directive will be applied to each element within the TABLE list. We also move the newline & leading spaces inside the `~{`.

```
> (format nil "~:{~&   ~6D  ~8@A  ~8@A~}" table)
"        1     GREEN    RIBBIT
       21    YELLOW      BUZZ
       17    ORANGE      MEOW
       29    PURPLE   SILENCE"
```

I'd like a good technique for printing the column headers. It's easy to print the words at the tops of the columns because the `~D` directive is forgiving of arguments that are not numbers, but I don't know a good way to print the hyphens at the tops of the columns.

You could print the words like this:

```
> (format nil "~:{~&   ~6D  ~8@A  ~8@A~}"
          (cons
            (list 'number 'color 'noise)
            table))
"   NUMBER     COLOR     NOISE
        1     GREEN    RIBBIT
       21    YELLOW      BUZZ
       17    ORANGE      MEOW
       29    PURPLE   SILENCE"
```

I can think of two techniques for printing the hyphens. In the first, you would construct strings of hypens & print them the same as you print the words at the tops of the columns. In the second technique, you would modify the format string so that the fill character was the hyphen character, then print empty strings. Neither technique is entirely satisfactory, so I'm still looking.

5 How to Use Format for Word Wrap

This trick is pretty damned cool for a function that formats output.

Let's pretend that a program has a list of words to print. Maybe they are the words from an error message. The program needs to insert newline characters so that the text fills the lines on the user's screen but no single word spans multiple lines.

In other words, you have a list of words like this:

```
> (setq words (list "There" "is" "no" "way"
                    "on" "god's" "green" "earth"
                    "I" "can" "perform" "that"
                    "function," "Will" "Robinson."))
```

The screen has only forty columns (maybe it's an old Atari 800), so you want the output to look like this:

```
There is no way on god's green earth I
can perform that function, Will
Robinson.
```

Lisp's FORMAT function can do this for you. In fact, it's easy (but tricky).

```
> (format nil "~{~<~%~1,40:;~A~> ~}" words)
"There is no way on god's green earth I
can perform that function, Will
Robinson. "
```

Here's how it works:

1. The `~{` applies its inner format string to each element in its argument. In our case, the inner format string is "`~<~%~1,40:;~A~> `". The argument is WORDS, which is a list of strings.
2. The inner string is applied to the next element in WORDS.
   (a) The "`~<~%~1,40:;~A~> `" has two clauses: "`~%~1,40:`" and "`~A`".
   (b) That first clause is special because it ends with two integers & a colon. That special type of clause means "Do not print this clause unless the length of the result of formatting the other clauses is longer than $40 - 1 \rightarrow 39$".
3. Until WORDS is the empty list, repeat step number 2.

Step 2.2 is where the action happens. The `~<` directive is given one of the words, which it formats according to its second clause, the `~A`. It takes the result of that operation, adds its length to the line position of the output stream (or the output string, since we're using NIL as FORMAT's first argument), & compares that sum to $40-1 \rightarrow 39$, which it gets from the first clause. If the sum is greater than 39, the `~<` outputs the first part of its first clause & resets the line position to 0. Then it outputs the word that it just formatted.

In an earlier version of this trick, I used `~&` instead of `~%`. That worked on clisp (tested it myself), but not on SBCL 0.8.8 (tested it myself), CMUCL (so I hear), or Allegro (so I hear). With a little research, it turned out that it works on clisp & SBCL with `~%` instead of `~&`. I presume it works on CMUCL & Allegro, too.

6 How to Print a Comma-Separated List

Sometimes you have a list of things, & you want to print it with comma separators. Here's a brute-force way of doing it:

```
> (defun comma-list (lst)
    (let ((str (format nil "~A" (first lst))))
      (dolist (x (rest lst))
        (setq str (format nil "~A, ~A" str x)))
      str))
COMMA-LIST
> (comma-list '(1 2 3 4))
"1, 2, 3, 4"
```

It works, but it's brute-force & explicit. Not nice.

We can make FORMAT do the work for us.

```
> (defun comma-list (lst)
    (format nil "~{~A~#[~:;, ~]~}" lst))
COMMA-LIST
> (comma-list '(1 2 3 4))
"1, 2, 3, 4"
```

The `~{` loops over the items in the list. Nothing new there. For each item, the `~A` prints it. Then the `~[` directive goes to work. The `#` evaluates to the number of remaining arguments to the `~[`. The `~[` has two clauses, & the last is prefaced with `~:;`, which causes it to be chosen if the numeric argument (the `#`) doesn't match a clause. So if therea re no more items to print, the first clause, which is the empty string, is selected. Otherwise, the second/last clause, which is a space & a comma, is selected. Magically, FORMAT produces a list of items that are properly separated by commas.

It's not a list of English because the last item is not prefaced by "and". (I'll do that later, in Section 7.)

The format string could be shortened even more with the `~^` directive, like this:

```
> (defun comma-list (lst)
    (format nil "~{~A~^, ~}" lst))
COMMA-LIST
> (comma-list '(1 2 3 4))
"1, 2, 3, 4"
```

The `~^` shortens the format string, but it breaks the "single entry, single exit" heuristic. Sure, few programmers give a damn about single entry & single exit these days, but I do & I won't change my views, so I don't like `~^`.

When is this useful? Many of my Lisp programs produce C programs. The C code frequently has long lists of forward declarations, like this:

```
/* example C code generated by Lisp */
extern struct Sometype S_283, S_284, S_2, S_17, S_9,
     S_11, S_12, S_21, S_33;
```

The comma-separated list trick of FORMAT is useful for that.

It combines with the word-wrap trick to make pretty code, too. Like this:

```
> (setq c-syms '(S_283 S_284 S_2 S_17
                 S_9 S_11 S_12 S_21 S_33)
       x nil)
NIL
> (format nil
          (concatenate 'string
            "extern struct Sometype "
            "~{~<~&     ~1,50:;~A~#[~:;, ~]~>~}"
            ";")
          c-syms)
"extern struct Sometype S_283, S_284, S_2, S_17,
     S_9, S_11, S_12, S_21, S_33;"
```

You might want to note that the format string from this example is indisputably an eye-sore.

7 How to Use Commas in English Style

If I have a list like this one:

```
> (setq lst (list 'red 'green 'blue))
(RED GREEN BLUE)
```

The proper form of list for an English reader would be red, green, & blue. FORMAT can do the work.

```
> (format nil "I have~{ ~(~A~)~#[~;, and~:;,~]~}." lst)
"I have red, green, and blue."
```

Or in Gene style:

```
> (format nil "I have~{ ~(~A~)~#[~;, &~:;,~]~}." lst)
"I have red, green, & blue."
```

8 Conclusion

FORMAT is comparable to but even more capable than C's venerable `printf`. I've shown some cool things you can do with FORMAT. I'm looking for even more.

A Change Log

2004-Oct-29
- A reader reported that the technique for word wrap (Section 5) doesn't work for him for CMUCL or Allegro. I confirmed that, then figured out that changing `~&` to `~%` in the word wrap example makes it work for SBCL as well as clisp. I presume it works for CMUCL & Allegro, though I have not tested it on those systems myself.
- At least the technique for printing a comma-separated list (Section 6) works with SBCL.
- Corrected a couple of spelling errors.

B Other File Formats

This document is available in multi-file HTML format at http://lisp-p.org/fmt/.

This document is available in DVI format at http://lisp-p.org/fmt/fmt.dvi.

This document is available in PostScript format at http://lisp-p.org/fmt/fmt.ps.

This document is available in Pointless Document Format (PDF) at http://lisp-p.org/fmt/fmt.pdf.

References

[Gra96] Paul Graham. *ANSI Common Lisp.* Prentice Hall, Englewood Cliffs, New Jersey 074623, USA, 1996. ISBN 0-13-370875-6.

[Inc88] Franz Inc. *Common Lisp the Reference.* Franz Inc., 1988. ISBN 0-201-11458-5.

[Wat89] Richard C. Waters. XP — A common LISP pretty printing system. Technical Report A.I. MEMO 1102a, Massachusettes Institute of Technology, Cambridge, Massachusetts, 1989. http://citeseer.ist.psu.edu/527352.html.

[X3J] ANSI Committee X3J13. Common lisp hyperspec. Xanalys Web site. http://www.lispworks.com/reference/HyperSpec/.