Clarification Proposal for CLHS Section 22.3

Didier E Verna

Clarification Proposal for CLHS Section 22.3

Version 1.1 – CDR 7

Didier Verna <didier@lrde.epita.fr>

Copying

1 Motivation

Section 22.3 “Formatted Output” of the Common Lisp Hyperspec describes the syntax and semantics of `format` directives. We believe that the standard is underspecified in two related areas.

1.1 Trailing Commas

The standard describes the syntax of directive parameters as follows.

> *A directive consists of a tilde, optional prefix parameters separated by commas, [...].*

It also gives the following example.

```
"~,+4S"     ;Here the first prefix parameter is omitted and takes
            ; on its default value, while the second parameter is 4.
```

Because this specification only mentions the *separation* of directive parameters from each other, it does not address the case where a final parameter would be followed by a comma (what we call a “trailing comma”). For example, consider the case of `"~+4,S"`. We could consider that the comma separates the parameter from the directive character, but since it is actually not needed in that case, we could also consider that there is a second, empty parameter after the first one. The standard does not allow us to decide on the proper interpretation of that directive.

1.2 Final V Parameters

As a corollary to this problem, a second one comes from the definition of the `V` parameter. Its association with a `nil` argument is described as follows.

> *If the arg used by a V parameter is nil, the effect is as if the parameter had been omitted.*

This is problematic when the `V` parameter is in final position. Indeed, “omitting” the parameter does not tell us what to do with the surrounding commas. In other words, assuming that a trailing comma makes a difference, would `"~+4,VS"` be equivalent to `"~+4,S"`, `"~+4S"` or even to `"~+4,,S"`? Would `"~+4,V,S"` be equivalent to `"~+4,,S"` or `"~+4,S"` or `"~+4S"`? Here again, the standard does not allow us to decide on the proper interpretation.

2 Analysis

In order to analyze the effects of these underspecifications, some tests were conducted on 8 implementations of Common Lisp on January 27th 2011, with the help of the user-defined `format` function below.

```
(defun fmt (stream argument colonp atsignp &rest params)
  (declare (ignore stream argument colonp atsignp))
  (format t "~S~%" params))
```

2.1 Trailing Comma Tests

We can test the effects of a trailing comma with the following `format` calls.

```
(format t "~1,2/fmt/" t)
(format t "~1,2,/fmt/" t)
(format t "~1,2:/fmt/" t)
(format t "~1,2,:/fmt/" t)
```

The results are as follows:

```
CMU-CL / CCL / CLISP / Allegro / LispWorks:
=> (1 2)
=> (1 2)
=> (1 2)
=> (1 2)

SBCL / ECL / ABCL:
=> (1 2)
=> (1 2 NIL)
=> (1 2)
=> (1 2)
```

This test exhibits some divergence that could turn out to be a portability problem for directives relying on the number of parameters they are given (including the '`/`' directive calling a user-defined function) and/or for the ones which don't default their parameters to `nil`.

CMU-CL, CCL, CLISP, Allegro and LispWorks behave in a consistent way which is to ignore a trailing comma, or, to put it differently, simply consider that a comma *ends* a parameter.

SBCL, ECL and ABCL, on the other hand, behave less consistently because the trailing comma is ignored only in the presence of a colon modifier (or an at-sign modifier, for that matter). We don't see any rationale for this specificity. It seems that if the policy is to take a trailing comma as an indication of a subsequent empty parameter, then the fourth test should give the same result as the second one.

2.2 Final V Parameter Tests

We can test the effects of a final `V` parameter with the following `format` calls.

```
(format t "~1,v/fmt/" t t)
(format t "~1,v/fmt/" nil t)
(format t "~1,v,/fmt/" t t)
(format t "~1,v,/fmt/" nil t)
(format t "~1,v:/fmt/" t t)
(format t "~1,v:/fmt/" nil t)
(format t "~1,v,:/fmt/" t t)
(format t "~1,v,:/fmt/" nil t)
```

The results are as follows:

```
CMU-CL / CCL / CLISP / Allegro / LispWorks:
=> (1 T)
=> (1 NIL)
=> (1 T)
=> (1 NIL)
=> (1 T)
=> (1 NIL)
=> (1 T)
=> (1 NIL)

SBCL / ABCL:
=> (1 T)
=> (1 NIL)
=> (1 T NIL)
=> (1 NIL NIL)
=> (1 T)
=> (1 NIL)
=> (1 T)
=> (1 NIL)

ECL:
(1 T)
(1)
(1 T NIL)
(1 NIL)
(1 T)
(1)
(1 T)
(1)
```

Here again, the divergence is to be expected, although some additional complication seem to appear. Indeed, different implementations exhibit different interpretations of what "omitted" means, and add more inconsistency to the results.

CMU-CL, CCL, CLISP, Allegro and LispWorks remain consistent with themselves, in the sense that they do consider commas as part of the preceding parameter. They also appear to conform to the Hyperspec example presented in Section 1.1 [Trailing Commas], page 2, where the word "omitted" really means that the parameter is *empty* in the `format` string. Thus, when associated with a `nil` argument, the directive `"~1,v/fmt/"` becomes equivalent to `"~1,,/fmt/"`, *not* to `"~1,/fmt/"` or even `"~1/fmt/"` which would both print as `(1)`. Note the second comma which serves as a delimiter for an empty, final parameter.

SBCL and ABCL also remain consistent with themselves, by considering trailing commas as an indication of a subsequent empty, final parameter. They also still exhibit the same difference

in behavior according to whether a modifier is added before the directive character, which continues to appear somewhat odd.

ECL, on the other hand, exhibits yet another interpretation of tests 2, 6 and 8. Apparently, a `V` parameter associated with a `nil` argument acts as if the parameter had not been specified *at all* in the `format` string. Thus for instance, `"~1,v/fmt/"` is considered equivalent to `"~1/fmt/"`, *not* to `"~1,,/fmt/"`. This also happens to be the case for non-final `V` parameters, as the following examples demonstrate.

```
(format t "~1,v,2/fmt/" nil t)
=> (1 2) ;; not (1 NIL 2)

(format t "~v,1/fmt/" nil 1)
=> (1) ;; not (NIL 1)
```

This behavior seems to be in contradiction with the apparently intended meaning of the word "omitted" in the standard, which, again, sounds like "empty" more than like "absent". More confusion arise when you note that built-in directives behave differently:

```
(format t "~+4@S" 1)
=>     1

(format t "~v,+4@S" nil 1)
=> 1
```

According to the previous examples, one would have expected to get the same output from both cases here. As a matter of fact, this divergence brings ECL closer to the other implementations as this time, the first parameter is "empty" rather than "absent".

3 Proposal

Given the preceding observations, we think that section 22.3 of the Common Lisp Hyperspec needs clarification on:

- the status of commas, either trailing ones or as a separator between two parameters,
- the exact semantics of the `V` parameter when associated with a `nil` argument, and the exact meaning of "omitted" in such a case.

3.1 Rationale for Commas

A very pertinent argument in favor or *not* giving a specific status to trailing commas is provided by Martin Simmons: it allows one to provide a single, empty parameter to a directive, something which would not be possible otherwise. Indeed, consider the following situation:

```
(format t "~,/fmt/")
=> (NIL NIL) ;; for SBCL
=> (NIL)     ;; for CMU-CL
```

There is no way SBCL can get a single, empty parameter with its current behavior, which can be problematic. Therefore it seems preferable to consider commas, including trailing ones, as "part of" the preceding parameter, and not give trailing commas any particular meaning. If one wants two empty parameters, one would just use two consecutive commas.

Since there is also no reason for trailing commas to behave differently, according to whether they are followed by the directive character or by a modifier, we therefore suggest that "conformant" implementations should behave like CMU-CL, CCL, CLISP, Allegro and LispWorks.

3.2 Rationale for V Parameters

As mentioned previously, the example of an "omitted" parameter in the Hyperspec really is that of an "empty" parameter, rather than that of an "absent" one (the comma separating it from the next parameter is still here).

CMU-CL, CCL, CLISP, Allegro and LispWorks interpret it this way. SBCL and ABCL also interpret it this way, but only when a modifier is present, which again, is not very consistent. ECL interprets "omitted" as "absent" for user-defined functions, but seems to interpret it as "empty" for built-in directives, which is not very consistent either.

Therefore, we suggest that the intended meaning of the word "omitted" in the standard really is "empty" instead of "absent", and again, that "conformant" implementations should behave like CMU-CL, CCL, CLISP, Allegro and LispWorks.

3.3 Summary

To summarize, we propose to clarify section 22.3 of the Hyperspec as follows, and we suggest that Common Lisp implementations conform to the current behavior of CMU-CL, CCL, CLISP, Allegro and LispWorks.

```
Prefix parameters are followed by a comma which acts as a separator
between the parameter and the next directive component (whether another
parameter, a modifier or the directive character itself). If the
parameter is the final one (that is, if the next directive component is
a modifier or the directive character itself) and if the parameter is
not empty, then the comma may be omitted.

For instance, the following directives are all equivalent:
~1S
~1,S
~1:S
~1,:S

Note that a comma is always needed to specify an empty parameter:
~,S   => one empty parameter
~1,,S => one "1" parameter and one empty parameter
```

```
If the argument used by a V parameter is nil, then the effect is as if
the parameter were empty.

For instance, and given the clarification above, the following
equivalences hold when the V parameter is associated with a nil
argument:
~v,1S   <=> ~,1S    ;; not equivalent to ~1S
~1,v,2S <=> ~1,,2S  ;; not equivalent to ~1,2S
~vS     <=> ~,S     ;; not equivalent to ~S
~1,vS   <=> ~1,,S   ;; not equivalent to ~1,S
```