

## Re: Report enhancements

**Source:** <http://coding.derkeiler.com/Archive/Cobol/comp.lang.cobol/2004-11/0056.html>

---

**From:** William M. Klein ([wmklein\\_at\\_nospam.netcom.com](mailto:wmklein_at_nospam.netcom.com))

**Date:** 11/02/04

Date: Mon, 01 Nov 2004 23:57:14 GMT

Chuck,

Robert has a good point here. I know that the '02 (and earlier) Standards REQUIRES that a "format 1" EXIT statement (the one that is just a "marker" for an exit-paragraph/section) must be the only statement in a sentence and that must be the only sentence in the procedure. HOWEVER, I think there are some implementations that don't require this as an extension. For those "adding" an EXIT PERFORM format would be a syntactic problem. Consider (indenting for clarity). I also KNOW that some implementations allow the "END-PERFORM" to be omitted at the end of an inline perform – before a period (similar to other scope delimiters).

Para1.

Display "Here"

Exit

Perform

Display "Here too"

.

\*\*\*

Clearly, this is NOT valid for the '02 (or '85 – or any other STANDARD compiler). However, I could imagine it being "confusing" to both compiler and programmer if coded as:

Para1.

Display "Here"

Exit Perform

Display "Here too"

.

--

Bill Klein

[wmklein <at> ix.netcom.com](mailto:wmklein@ix.netcom.com)

"Chuck Stevens" <[charles.stevens@unisys.com](mailto:charles.stevens@unisys.com)> wrote in message  
news:[clrqli\\$bda\\$1@si05.rsvl.unisys.com](mailto:clrqli$bda$1@si05.rsvl.unisys.com)...

> Robert Wagner wrote:

>

>> I think the reason is syntactical. EXIT FOO can be read GO TO

>> (nearest) END-FOO (and stop looping for perform), where END-FOO can be

>> explicit or implicit.

## comp.lang.cobol: Re: Report enhancements

```
>
> Not sure what this means; could you clarify?
>
>> An out-of-line EXIT PERFORM cannot ascertain
>> where the exit is by syntax alone; it must get the address from the
>> stack at execution time.
>
> This I understand. The object code has a problem figuring out whether the
> statement is executed under control of a PERFORM by examining the stack at
> execution time ... exactly why?
>
> All our compilers need to generate for out-of-line EXIT PERFORM is DUPL,
> ZERO, GRTR, BRFL (continue), DLET, DBUN. If what's on the top of the stack
> at the start of *any* statement is a zero, we're not in an active
> out-of-line PERFORM (unless the stack is munged, in which case we're hosed
> anyway); if it isn't, we are. Looks really simple to me.
>
>> You are using semantics rather than syntax. To this outside observer,
>> it seems J4 would not be receptive.
>
>
> Not sure I understand why not. EXIT PERFORM is *syntactically* disallowed
> outside the context of an *inloine* PERFORM; I'm not convinced the
> restriction is appropriate. The proposed *semantics* of EXIT PERFORM would
> be exactly analogous to those of EXIT PROGRAM when the program in question
> wasn't CALLED -- CONTINUE. I don't see what it'd *break* to provide it, and
> I can certainly see the utility (of both EXIT PERFORM and EXIT PERFORM
> CYCLE) in the context of an out-of-line PERFORM, not just for clarity of
> syntax but for functionality otherwise unavailable (except for GO TO).
>
>> >I agree that an out-of-line indefinite PERFORM pretty much requires
> either
>> >the despised GO TO or the prohibited EXIT PERFORM, but I'm not really
>> >opposed to relaxing the restriction on the latter, unless someone
> convinces
>> >me that there's a good reason why it's there.
>>
>> I can't think of a good reason.
>
> Well, there ya go. That's my point.
>
>> But it would require TRUE and FALSE to be boolean literals, as they
>> are in most other languages and in Micro Focus as an extension.
>
> No, not quite right. The syntax rules can permit their use in specific
> contexts.
>
>
>> While we're on the subject, it seems asymmetrical that we can say SET
>> CONDITION-1 TO FALSE but cannot say
>>
>> 01 PIC X VALUE FALSE.
>> 88 CONDITION-1 VALUES 'A' 'B' 'C' FALSE 'D'.
>
> I think you've misunderstood. VALUE FALSE applies *only* to 88's, and the
> filler isn't an 88.
>
> I see no inconsistency. *ALL* the values save 'A', 'B' and 'C' would result
> in CONDITION-1 being FALSE; there may be any number of FALSE values. There
> may be only one such value to which the parent item is set *when* you SET
> the condition to FALSE. "VALUE FALSE" is unnecessary as well as
> meaningless on the PIC X item.
```

## comp.lang.cobol: Re: Report enhancements

```
>
>> When the value of false changes to 'E', we must change 01 VALUE as
>> well as 88 VALUES. That's contrary to the spirit of 88s.
>
> 'E' is but one of probably 253 values for the PIC X item that would cause
> CONDITION-1 to be FALSE. VALUE FALSE has nothing to do with the value that
> RESULTS in FALSE, it only specifies which value is used when it is
> EXPLICITLY SET to FALSE.
>
>> Examples follow. Please explain how we can do this with '02 features.
>
> Easy.
>
> Method 1: Use >>DEFINE I-AM-DEBUGGING at the beginning of the program and
>>>IF I-AM-DEBUGGING... >> END-IF pairs around the IFs and END-IFs. Maybe
> not quite as elegant as your examples, but functional.
>
> Method 2:
>
> D77 PIC 9 VALUE 1.
> D88 ITS-TRUE VALUE 1.
> D88 ITS-FALSE VALUE 0.
>
> D OR ITS-TRUE ...
> D AND ITS-FALSE ...
>
> Oh. Wait. That works in COBOL85. To get it up to COBOL2002 I'd have to do
> something like USAGE BIT ...
>
> -Chuck Stevens
>
>
```