IEEE Standards Interpretations for IEEE Std 1003.2™-1992 IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX®)--Part 2: Shell and Utilities

IEEE Standards Department, Copyrights and Permissions, 445 Hoes Lane, Piscataway, New Jersey 08855-1331, USA

Interpretation Request #107
Topic: various awk requests **Relevant Clauses:** 4.1

Number 1 of 23: page 163, subclause 4.1.2./page 172, subclause 4.1.7.3.
Lines 21 and 22 describe what happens if $0 is updated, that NF is changed to reflect this. Unfortunately, nothing is said about what happens when NF is assigned to. I suggest adding the following wording to 4.1.2. Assigning a new value to NF changes the value of $0. If the new value is smaller than the previous one, then the extra fields are deleted. If the new value is greater than the previous one, the new fields are created with the null string as their value, and the record is rebuilt using the current value of OFS to separate the fields. This could be added to 4.1.7.3 instead.

Number 2 of 23: page 167, subclause 4.1.7.1.
Lines 143-145 state that”a missing action shall be equivalent to an action that writes the matched record of input to standard output.” The traditional meaning of a missing action has been that it is equivalent to ‘{ print }’. This has important semantic ramifications, in that the value of ORS can affect how the record is written to standard output. The current wording could conceivably treat a missing action as if it were { printf “%s”, $0 } with no trailing record separator. I suggest adding”, as if by a call to the print function with no arguments” to the end of line 145.

Number 3 of 23: page 167, subclause 4.1.7.1.
lines 361 to 364 state that NF retains its value inside an END action. But no mention is made anywhere as to whether $0 keeps the value of the last record read, inside the END action. Unix awk sets $0 to “” and NF to 0 inside an END action. GNU awk and mawk both preserve $0 and NF into the END action. In e-mail to me, I have been told that logically it makes sense for $0 and NF to be preserved in the END rule, and that it just

wasn't done because it would have complicated the implementation. Since POSIX already says that NF is preserved, it should say that $0 is preserved too. Add the following text to the end of 4.1.7.1, on line 155, or as a new paragraph. Within an END action, $0 shall refer to the value of the last record read from the last input file. (Of course, within an END action, getline commands with a redirection and without a variable shall change the value of $0 and NF.)

Number 4 of 23: page 168, subclause 4.1.7.1.
No mention is made of what the behaviour should be for an empty source program: awk '' file1 ... or if the source file is empty awk -f source data ... and the source file is of zero size. All current implementations simply exit silently with an exit value of zero. At the beginning of the section, after line 138, insert the following text: If either the source code operand on the command line, or the source file(s) supplied via the -f option are empty, awk shall immediately exit with a return status of zero. Gawk is the only awk that will split strings across newlines using \, like C. This would also be a useful feature to standardize, making awk more like C.

Number 5 of 23: page 170, subclause 4.1.7.2.
Lines 274-297 describe arrays. Nowhere is it stated that arrays and scalars share the same name space. On line 279, right before the description of how unsubscripted array names can be used, add the following sentence. An array and a scalar variable cannot have the same name. (Actually, this is noted on lines 750 and 751, when discussing parameter names; that is an unobvious place, and the information should be duplicated here too.)

Number 6 of 23: page 170, subclause 4.1.7.2.
Lines 298-303 describe the rules for comparisons. Unfortunately, they are just plain broken, because of they way numeric strings are compared as numbers. The current rules cause the following code if (0 =="000") print"strange, but true" else print"not true" to do a numeric comparison, causing the if to succeed. It should be intuitively obvious that this is incorrect behaviour, and indeed, no existing implementation of awk actually behaves this way. The solution is a bit involved. First, at line 269, add the statement This type is used when performing comparisons. Next, replace lines 298-303 with a new subclause describing how comparisons are done.

Here is the text we will be using in the GNU Awk manual:

1. A numeric literal or the result of a numeric operation has the NUMERIC attribute.
2. A string literal or the result of a string operation has the STRING attribute.
3. Fields, getline input, FILENAME, ARGV elements, ENVIRON elements and the elements of an array created by split that are numeric strings have the STRNUM attribute. Otherwise, they have the STRING attribute. Uninitialized variables also have the STRNUM attribute.
4. Attributes propagate across assignments, but are not changed by any use. When two operands are compared, either string comparison or numeric comparison may be used,

depending on the attributes of the operands, according to the following (symmetric) matrix:

```
+-------------------------------------------- |    STRING     NUMERIC
STRNUM
--------+--------------------------------------------
|
STRING |    string string string
|
NUMERIC | string numeric      numeric
|
STRNUM |   string numeric      numeric
--------+--------------------------------------------
```

This is the single biggest problem in the awk part of the standard. It is critical that the comparison rules be fixed. The above text is what both the GNU Awk developers and the mawk author , have agreed to and have been using for around three years. It is correct, and it works.

Number 7 of 23: page 171, subclause 4.1.7.3.
Lines 313 and 314 state that references to non-existent fields produce the null string. It should also be stated that no new field is created. Add the following sentence: Such references do not actually create new fields beyond $NF.

Number 8 of 23: page 171, subclause 4.1.7.3.
Lines 335 and 336 discuss the CONVFMT variable. It should be reiterated that the result of a number to string conversion is unspecified if CONVFMT is not a floating point format. Copy the statement to this effect from lines 179-181 to this point.

Number 9 of 23: page 172, subclause 4.1.7.3.
Line 372; the word “separation” should be “separator”. Make the change.

Number 10 of 23: page 173, subclause 4.1.7.4.
Lines 394 and 395 state that a / must be escaped with a \ inside an ERE. This is really only true for ERE tokens, not for string constants used as EREs. Change the wording to: Using a slash character within an \f(CWERE\fP token requires the escaping shown in Table 4-2.

Number 11 of 23: page 173, subclause 4.1.7.4.
Lines 400-402 discuss the use of matching, and mention the circumflex and dollar operators. It needs to be emphasized that in awk, ^ and $ apply only to the beginning and end of the *string*, and that they don’t work against embedded newlines. Add the following after and outside the parenthesized remark. The circumflex and dollar-sign special operators apply to the string as a whole, and not to any newlines embedded in the string; the last character before an embedded newline is not matched by the dollar-sign,

nor is the first character after an embedded newline matched by the circumflex.

Number 12 of 23: page 173, subclause 4.1.7.4.
Lines 422-428 describe the behaviour of FS. They leave out the case where FS is the null string. In this case, Unix awk, GNU awk and mawk all treat the whole record as one field. In the next major release of GNU awk, this case will be used to split the record into fields that are a single character in size. Therefore, the case for FS = “” should be made explicitly unspecified (I’ve had agreement that single character splitting is a good idea). Renumber items (1) and (2) to be items (2) and (3). Insert the following new text for item (1): (1) If FS is the null string, the behaviour is unspecified.

Number 13 of 23: page 173, subclause 4.1.7.4.
Line 434 should state that include the ~ and !~ operators in the list of cases where matching is not based on input records. Change the sentence to Except for the ~ and !~ operators, and in the gsub, match, split and sub built-in functions, ...

Number 14 of 23: page 175, subclause 4.1.7.6.1./page 180, subclause 4.1.7.6.2.3.
Lines 509 and 716 both indicate that the expression is evaluated to produce a string to be used at the *full* pathname for the file to be opened. If I understand this correctly, this means a pathname beginning with a slash. This is both unnecessary and incorrect. I also don’t see the term”full pathname” defined in 2.2.2 (but only after a brief look). Remove the term”full” from both these lines. A careful search of the entire text of the awk standard should be performed for similar usages; these are the only two that I found but there may be others that escaped me.

Number 15 of 23: page 175, subclause 4.1.7.6.1.
There is an extraneous period after the right parenthesis on line 515. Remove the period. (:-)

Number 16 of 23: page 176, subclause 4.1.7.6.1.
Lines 539 and 540 say that the printf format is similar to that used in 2.12, and that’s it. What happens if someone uses a C style format, like ‘%lf’? It’s not specified at all. Either flags not listed in 2.12 should not be allowed, or the behaviour should be unspecified. Add the following after line 540: The format notation shall not contain flag characters other than those specified in clause 2.12. This makes the characters illegal. I think this is best, since they flags like ‘l’ and ‘h’ make no sense for double precision numbers. (Hmmm, is this covered by item 10 on page 177?)

Number 17 of 23: page 176, subclause 4.1.7.6.1.
There is no discussion of what the behaviour should be when an integer format is used (%d, %x, %o, etc) and the double precision number is outside the range of a long or unsigned long. GNU awk currently detects this and switches to %g notation, other awks do different (weird) things. Change the current item (10) on page 177 to (11), and insert the following text as the new item 10. (10) If the d, i, o, u, x, or X conversion specifications are used, but the value to be formatted is outside the range of long integer

(if negative) or unsigned long integer (if positive), then the g conversion shall be used instead. (References could be made to LONG_MIN and ULONG_MAX instead, if that is more suitable...) I prefer this over making it unspecified, since the users can rely on the output being sane, without having to worry about portability.

Number 18 of 23: page 177, subclause 4.1.7.6.2.1.
Line 592 does not state that the units of the x and y arguments to atan2 should be in radians, and that the return value is also in radians. Change line 592 to atan2(y,x) Return the arctangent of y/x, where x and y are in radians, and the result is in radians.

Number 19 of 23: page 178, subclause 4.1.7.6.2.2.
Lines 629-638 describe the split function. Not mentioned here is the fact that the array is cleared first before the new elements are assigned to it (this is existing practice). Here is some code showing the semantics: a[1] = 1; a[2] = 2; a[3] = 3 split("1 2", a) if (3 in a) print"awk is broken" else print"all is ok" I.e., all the elements are deleted; a[3] is not kept around just because only two elements were split out of the string. At line 630, after the first sentence, insert the following sentence: All elements of the array a are deleted from the array before the split is performed.

Number 20 of 23: page 179, subclause 4.1.7.6.2.2.
Lines 656-660 describe the substr function. It is not quite clear what happens if you supply a requested number of characters that is greater than the number of characters left in the string. On line 658, change "If n is missing" to "If n is missing, or if n specifies more characters than are left in the string,".

Number 21 of 23: page 179, subclause 4.1.7.6.2.3.
Line 693 uses the word "file" whereas the rest of the section on getline uses the word"stream". Change "file" to"stream".

Number 22 of 23: page 181, subclause 4.1.7.6.2.4.
The discussion of function arguments does not state anywhere that duplicate parameter names are not allowed. Add the following statement to the end of the paragraph at line 751. No parameter to a function shall have the same name as another parameter of the same function.

Number 23 of 23: Suggested Enhancements for AWK
The following three items are features available in at least one version of awk that could profitably be added to the awk standard. He recently added an 'fflush' function that takes the name of an open output file or pipe and flushes the output. GNU Awk picked this up and extended it so that a call with either a null string or no argument flushes all open files. A description for this might be flush([FILE]) Flush any buffered output for the file or pipe FILE. If FILE is omitted, buffered output for all open files and pipes is flushed. It would be really nice to have some way to mix command line source code and code from a file. This would allow the use of library functions together with programs on the command line. I'd like to suggest a '-s' option for this: awk -s 'my program' -f libfile1 -f

libfile2 datafile1 datafile2 ... Gawk uses a -W option for this, currently. Gawk has a variable ARGIND, that indicates the index in ARGV of the current data file being read. This is useful for tracking how far along in the data one is. Text for the description could be: ARGIND The index in ARGV of the current data file being processed. ARGIND may be changed by the user's program, but it is automatically reset whenever a new data file is opened.

Interpretation Response

#1: The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

#2: The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

#3: The standard clearly does not specify what happens to $0, and that NF does not change from the previous value, and conforming implementations must conform to this.

#4: The standard is clear in that it requires all input files to be read, and conforming implementations must conform to this. However, this does not match historic practice, and concerns have been raised about this which have been referred to the sponsor.

#5: The standard states that the same name shall not be used within the same scope of both a scalar variable and an array, and conforming implementations must conform to this. However, concerns have been raised about this which are being referred to the sponsor.

#6: The standard states the rules for comparison, and conforming implementations must conform to this. However, concerns have been raised about this which are being referred to the sponsor.

#7: We do not believe that there is any confusion generated by the standard on this issue, however, concerns have been raised about this which are being referred to the sponsor.

#8: The standard clearly states in line 320 of page 171 a reference to 4.1.7.2 concerning behavior for the CONVFMT variable, and conforming implementations must conform to this.

#9: This is an editorial change and is accepted without comment.

#10: The standard states the behavior for / and \, and conforming implementations must conform to this. However, concerns have been raised about this which are being referred to the sponsor.

#11: The standard clearly states in subclause 2.8.4 and 2.8.4.6 the behavior for ^ and $, and conforming implementations must conform to this.

#12: The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

#13: The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

#14: We do not believe that the term “full pathname” implies a path beginning with a “/”. This would be referred to as “absolute pathname”. However, we agree that the term “full pathname” is not defined by the standard. Therefore the normal english usage of the term “full” would mean that it is the entire pathname as opposed to part of a pathname, but the pathname could be either relative or absolute. However, concerns have been raised about this which are being referred to the sponsor.

#15: This is an editorial change and is accepted without comment.

#16: The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

#17: Subclause 4.1.7.6.1 page 177 line 572-575 indicate that values being printed except for characters are converted to the appropriate type for the conversion specification. For %d, the rules are described in 4.1.7.2 page 167, lines 173-184 as modified by page 176 lines 532-538 which indicates that numeric values that are exactly equal to the value of an integer are converted using a %d format. Otherwise using the format specified by the format OFMT(which by default is %.6g). Subclause 2.9.2.1 says that integer variables in awk shall be implemented as if it were a C signed long data type. Therefore a value overflowing a C signed long would not have a numeric value taht is equal to an integer. Therefore the standard requires that these values be printed using the OFMT format. The standard clearly states the behavior for printing and formatting, and conforming implementations must conform to this.

#18: The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

#19: The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

#20: The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

#21: This is an editorial change and is accepted without comment.

#22: The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. Further, we believe that it is obvious that application writers should not use this construct.

#23: The standard obviously does not implement the suggested enhancements, this interpretation request does not address any issue presented in IEEE 1003.2-1992. To add new functionality to a standard, a PAR should be submitted.

Rationale for Interpretation

None.

Interpretation Follow up

Number 1 of 9: page 167, subclause 4.1.7.1.

lines 361 to 364 state that NF retains its value inside an END action. But no mention is made anywhere as to whether $0 keeps the value of the last record read, inside the END action. Unix awk sets $0 to “” and NF to 0 inside an END action. GNU awk and mawk both preserve $0 and NF into the END action. In e-mail to me, I have been told that logically it makes sense for $0 and NF to be preserved in the END rule, and that it just wasn’t done because it would have complicated his implementation. since POSIX already says that NF is preserved, it should say that $0 is preserved too. Add the following text to the end of 4.1.7.1, on line 155, or as a new paragraph. Within an END action, $0 shall refer to the value of the last record read from the last input file. (Of course, within an END action, getline commands with a redirection and without a variable shall change the value of $0 and NF.)

The response was: The standard clearly does not specify what happens to $0, and that NF does not change from the previous value, and conforming implementations must conform to this. The answer merely restates the problem, which IS that the standard does not specify what happens to $0. I believe that it should! Therefore I propose the above suggested new wording.

Number 2 of 9: page 176, subclause 4.1.7.6.1.

There is no discussion of what the behaviour should be when an integer format is used (%d, %x, %o, etc) and the double precision number is outside the range of a long or unsigned long. GNU awk currently detects this and switches to %g notation, other awks do different (weird) things. Change the current item (10) on page 177 to (11), and insert the following text as the new item 10. (10) If the d, i, o, u, x, or X conversion specifications are used, but the value to be formatted is outside the range of long integer (if negative) or unsigned long integer (if positive), then the g conversion shall be used instead. (References could be made to LONG_MIN and ULONG_MAX instead, if that is more suitable...) I prefer this over making it unspecified, since the users can rely on the output being sane, without having to worry about portability.

The response was: Subclause 4.1.7.6.1 page 177 line 572-575 indicate that values being printed except for characters are converted to the appropriate type for the conversion specification. For %d, the rules are described in 4.1.7.2 page 167, lines 173-184 as modified by page 176 lines 532-538 which indicates that numeric values that are exactly equal to the value of an integer are converted using a %d format. Otherwise using the format specified by the format OFMT(which by default is %.6g). Subclause 2.9.2.1 says that integer variables in awk shall be implemented as if it were a C signed long data type. Therefore a value overflowing a C signed long would not have a numeric value that is equal to an integer. Therefore the standard requires that these values be printed using the OFMT format. The standard clearly states the behavior for printing and formatting, and conforming implementations must conform to this.

As well as I can interpret this answer, it is saying that the standard says When using print, and the number is an integer but outside the range of a signed long, OFMT should be used. While this is useful information to me, it does not answer the question, which was”what should be printed when using printf (note, NOT print), and an integer type format, for a number that is outside the range of a signed long?” The quoted sections refer to conversions between string and numbers, and do not appear (to me, anyway) to be addressing the issue of *formatting*. Therefore, I again propose the above wording. (Upon re-reading my original question, it becomes clear that I was not explicit enough in the question, which related to printf/sprintf. Thus I am partially at fault for not”asking the right question”. I hope that the problem is now more evident.)

PASC Response to Resubmission
Part 1: The IEEE interpretation process does not change the standard, it merely interprets what the standard states. Because the standard is silent it does not place any requirements on $0 within an END action, therefore, the standard allows both behaviors provided by gnu awk, and unix awk.
Part 2: The standard is silent on what happens when a value to be printed by printf overflows a long data type, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.