Date: 31 October 1998
To: J3
From: Van Snyder
Subject: Miscellaneous comments and questions about 98-007r3

Given the prohibition at [59:6-8] "A named data object shall not be explicitly specified to have a particular attribute more than once in a scoping unit," do we need the constraint at [60:3] "The same *attr-spec* shall not appear more than once in a given *type-declaration-stmt*?"

At [59:26] we appear to prohibit parameterized polymorphic objects. Is this the intent?

In 5.1.2.3 [65-66] and 12.4.1.2 [241:12-15], do we need to say the same things about ALLOCATABLE that we say about POINTER?

In 5.1.2.7 on page 70, we remark that a pointer shall be associated or allocated before it's accessed. We don't say anything parallel about allocatable arrays in 5.1.2.9 on page 71. Do we say it anywhere? It would be clearer if we consistently said these things about pointer and allocatable in the descriptions of the attributes, or consistently said them elsewhere.

At [72:19] "Forran" ⇒ "Fortran".

At [72:19] "Responsability" ⇒ "Responsibility".

At [129:18-22] the only reasonable interpretation is that the kind type parameters of the type of *target* that are inherited from the type of *pointer-object* shall have the same values. It may take modifications at more than this point to do the best job of describing this.

At [162:31] the PAD= specifier can be specified in an OPEN statement, but not in a READ statement. It therefore can't be specified for standard input. The most efficient way to discover the length of an input record is to specify PAD="no", SIZE=*var* and use non-advancing input. This can't be done for standard input. (This is a very old problem, not introduced in 98-007r3.)

At [162:33-34, 165:11-15, 165:20-4] default modes for ROUND= and DECIMAL= specifiers cannot be specified for standard input and standard output. Therefore, the likely most common usage of these specifiers will be to put them into READ and WRITE statements. The purpose of having these specifiers in OPEN, namely so as not to need to specify them in every READ and WRITE statement, appears to have been compromised. If a few non-positive unit numbers were standardized (see remarks at [181:19] below), it would be possible to re-open standard input and standard output for the purpose of specifying these modes.

At [165:38] "he" ⇒ "the".

At [167:37] the ROUND= specifier cannot be specified in the control information list in a data transfer statement. Therefore one must specify rounding in the OPEN statement or within the format. The former cannot be done for standard input or standard output, and the latter cannot be done for list-directed or namelist formatting. Therefore, it is completely impossible to specify rounding modes for data transfer to standard input and standard output when using list-directed or namelist formatting. If a few non-positive unit numbers were standardized (see remarks at [181:19] below), it would be possible to re-open standard input and standard output for the purpose of specifying the rounding mode. It should also be made possible to specify the default (pre-format-examination) rounding mode in control lists.

At [168:16-17] I just noticed that ADV= can't be specified for internal files. This wasn't introduced in 98-007r3, but that doesn't make it a less silly restriction.

At [168:26-27] I just noticed that SIZE= can only be specified in an input statement that contains an ADV= specifier with the value NO. This wasn't introduced in 98-007r3, but that doesn't make

it a less silly restriction.

At [169:38] add "(9.4.3)" at the end of the sentence.

At [171:17] isn't the usual terminology "specifier with a value of" instead of "specifier of"?

At [173:5-8] additional work is required to allow for the case that a derived type object having pointer components is processed by derived-type input/output routines (see 173:24-26). Maybe just remove the sentence.

At [174:3] perhaps "not equivalent" should be "not necessarily equivalent".

At [176:33-34] the caveat "when execution of the statement begins" begs the question whether it is allowed for a user-defined derived-type input/output routine to close or reopen the file. There appears not to be a prohibition in 9.4.4.4.3. Should this be "throughout execution"?

There is no INTERFACE(IOLENGTH) specification to discover the length of records written at least in part by user-defined derived-type input/output routines. An argument was presented that such is not necessary, as one can simply put the appropriate list items in an INQUIRE by IOLENGTH, the processor will pretend to do output, but the processor's basic routines can notice that INQUIRE by IOLENGTH is in progress and suppress actual data transfers. Unfortunately, 9.4.4.4.3 is written in terms of "data transfer input/output statements" which therefore excludes the possibility to access user-defined derived-type input/output procedures from the output list in an INQUIRE by IOLENGTH statement.

At [181:19] the unit number is a "processor dependent negative value". Every time I see "processor dependent..." I wonder whether we're too frightened, too stupid or too lazy to standardize, or there really is a good reason not to standardize. In this case, there's no good reason not to standardize. One argument that has been advanced is that some vendors' libraries might use negative numbers internally. This is such a lame argument it *must* be a cover for something else nobody wants to talk about. Nobody admits that *his* library does this, but we shouldn't standardize non-positive unit numbers because *somebody else's* library *maybe* uses them. Even if *anybody's* library does this, changing to avoid using standard non-positive unit numbers can't possibly amount to more than 0.01% of the work involved in implementing user-defined derived-type I/O, and when we throw in all the rest of the changes in Fortran 2000, the necessary work nearly vanishes in comparison (if it exists at all in the first place). Allowing access to unit numbers for standard input and standard output would alleviate problems noticed elsewhere in this paper – e.g. one can't specify the default decimal, rounding and pad modes for standard input and output – and would ease programming of many applications. I suggest $0 \Rightarrow$ null (output doesn't do anything, and input always results in detecting end-of-file), $-1 \Rightarrow$ standard input, $-2 \Rightarrow$ standard output, $-3 \Rightarrow$ standard error if the system supports the concept else standard output, and any other non-positive value is an error. (Except maybe -4 could be standardized for the case of pseudo-output triggered by INQUIRE by IOLENGTH.) Example: Applications frequently contain such things as

```
IF ( MYOUT < 0 ) THEN
  WRITE (*,10) <giant-output-list>
ELSE IF ( MYOUT > 0 ) THEN
  WRITE (MYOUT,10) <same-giant-output-list-as-above>
END IF
```

Output lists can't be actual arguments, so if one wants to hide this in a procedure, one needs a separate procedure for each <*giant-output-list*>. Typically, such procedures will be called once, so there's no point to having them. It would be much cleaner to write

```
WRITE (MYOUT,10) <same-giant-output-list-as-above>
```

and depend on the behavior outlined above to achieve the desired effect.

The prohibitions against recursive data transfer statements on page 193 appear to be harder to enforce than to allow in the general case, given the necessity to allow them in order to support user-defined derived-type input/output. Recursive input/output statements should have exactly the same restrictions and privileges as user-defined derived-type input/output.

At [206:22-23] the note suggests that blanks are removed from the character literal in the DT edit descriptor, but I couldn't find normative text to support this. Should the note be corrected, or should normative text be added that specifies that blanks are removed from the character literal in the DT edit descriptor?

At [209:9] "output" → "input"?

At [209:43] "files" → "file".

At [210:1] "descriptor" → "descriptors".

In 10.6.8, 10.8 and 10.9, is there any special reason we can't just allow semicolons all the time for list element separators? The wording of these sections would be simpler. Vendors' I/O libraries would probably be simpler, too.

At [213:32, 213:46-1] is it necessary to define the value separator again? Why not just refer to or rely on the definition in 10.8.0?

At [211:40-46] it's OK to have blanks or end-of-record between the real and imaginary parts of a complex number, but not between the numbers and the parentheses. This wasn't introduced in 98-007r3, but that doesn't make it a less silly inconsistency.

At [239:34] "theh" ⇒ "the".

At [239:37] add "(4.5.1)" after "argument".

Throughout 12.4.1.2 [240-242], discussions of pointer or nonpointer probably need to be extended to include allocatable or nonallocatable. Do we need to address the question whether an allocatable dummy can be associated with a pointer actual, or vice-versa? If this is addressed elsewhere, it probably belongs in 12.4.1.2.

At [240:33] there is a reference to the "declared type of the actual argument." Concerning the description of the SELECT TYPE statement, Richard Maine questioned whether an expression has a "declared type." The same question applies here.

At [242:11] "may" ⇒ "shall".

In section 12.5, there is no discussion of what a dummy argument is and is used for. Should there at least be a reference to 12.4.1?

Should defined assignment be used for any of items 4, 5 or 10 in 14.7.7?