Public Review Comment #10 by Richard Maine

To: J3
From: Craig Dedo
Date: November 11, 2002
Subject: Public Review Comment #10 by Richard Maine

From: Richard Maine
Sent: Friday, November 1, 2002 6:47 PM
To: Donovan, Deborah
Subject: Comments on Fortran 2000 Committee Draft

The following are my public comments on the Fortran 2000 Committee Draft formally known as ISO/IEC CD 1539-1, Information technology - Programming languages - Fortran.

I might later add some more, but as the deadline is approaching, I am sending these to you now rather than waiting and possibly missing the deadline for them.

I have segregated these comments by category as typographical errors, wording problems, technical issues, and changes in the selection of features included in the language.

All references are to J3 paper numbers.

TYPOGRAPHICAL ERRORS

The following are fixes for typos, spelling errors, and similarly trivial errors in the CD.

[56:Note 4.50 last line] "END FUNCTION POINT_3D" ->
"END FUNCTION POINT_3D_LENGTH"
[202:28] specifer -> specifier
[350:9] depedent -> dependent
[377:Note 14.12] "LOGICAL MATRIX_ERROR =3D .FALSE." ->
"LOGICAL :: MATRIX_ERROR =3D .FALSE."
(One could argue for other colons in this and the subsequent note, but that is a style question. This one is an error).
Notes 10.11 and 10.12 have hyphens that should be minus signs.

WORDING PROBLEMS

W1. The term "binding" is used for two unrelated things, which is likely to be a cause of confusion. It is used in reference to type-bound procedures and in reference to C interoperability. I suggest that this confusion be resolved by using different terminology for C interoperability. Specifically, rename the BIND attribute and BIND statement to LANG (or LANGUAGE) (which I think will be more intuitively clear to most Fortran programmers anyway), rename the ISO_C_BINDING module to just ISO_C, and replace the term "binding label" with "linking label".

W2. The use of the word "predictable" in 2.5.5 is either confusing or arguably just plain incorrect and not supported by anything elsewhere in the draft. The only other place in the draft that any form of "predict" appears is in Note 12.26.

W3. The phrase "objects or subobjects" at [33:12] is nonsense insomuch as "Subobjects are themselves data objects" ([16:31]). The best fix is to just delete the sentence at [33:12] as it is just a poor attempt at rephrasing what is said correctly at [16:33-34] ("A subobject of a variable is a variable"). There is no need to repeat this on page 33, where it doesn't belong anyway.

Likewise, the phrase "object or subobject" at [114:24-25] should be replaced by just "object". The "or subobject" there is worse than useless; it is actively confusing.

W4. The term "literal constant" is defined only for source code; it is not something that appears in a formatted file. F90 interpretation 131, published in f90 corrigendum 2, fixed most of the inappropriate uses of this term. However, several remain in subclauses 10.9 and 10.10 of this CD. One example is at [239:27-28], where it says that c and r are literal constants, but then tries to rule out those parts of the syntax of literal constants that don't fit.

W5. The term "activation record" appears only once in the draft (in Note 7.9) and is never defined. It seems to me that this will just confuse those readers who don't know what an activation record is and will be obvious to those who do. I'd suggest deleting the sentence.

W6. The para at [102:18-23] uses the terms "starting point" and "ending point" inconsistently with their definition and use in the preceeding para. I suggest that the definitions be changed to include the word "value". Ref 02-276, 02-220r1.

W7. In the sentence at [3-4] is difficult to parse what "of the scoping unit" modifies. Ref 02-276, 02-221r2.

W8. The sentence at [399:37-39] apparently mentions three FORALL constructs, but leaves it confused as to what relationship among them is referred to, or even if it is actually 3 separate ones. Ref 02-276, 02-221r2.

W9. The sentence structures in [402:1-9] are so complicated that it is next to impossible to figure out what modifies what, and thus what these sentences actually mean unless you knew the answer before reading them. Ref 02-276, 02-221r2.

W10. There is substantial duplication between 2.4.6 and 5.1.2.11, with no obvious criterion for what is duplicated and what is not. The parts that are duplicated in 5.1.2.11 are insufficient to make it hold together without consulting 2.4.6, which makes one wonder why it doesn't just xref 2.4.6 and avoid the duplication. Ref 02-276, 02-240r1.

W11. The wording "in contexts where...a pointer refers to a target" [16:32-33] is vague and subject to any number of interpretations. Strangely, that wording was added here in the same J3 meeting at which the same wording in 6.1.2 was deleted for being too vague. The vagueness is, if anything, worse in 2.4.3.1, which lacks the context of 6.1.2. Ref 02-276, 02-241r4.

W12. The "and" on [19:5] should be an "or". Likewise on [418:26]. Ref 02-276, 02-241r4.

W13. The phrase "may be" is used twice in [19:23-25] where no permission is implied. Ref 02-276, 02-241r4.

W14. The word "components" at [41:7] is bolded as a definition, but this isn't a definition. Likewise for the word "finalized" at [59:19]. Ref 02-276, 02-242r2.

W15. Some of the "an"s in [114:11-12] should be "any"s. The present wording implies that there is always one and only one such argument, which is incorrect. There could be none or many. Ref 02-276, 02-242r2.

W16. The meaning of the phrase "used in any way" at 256:7 is vague. It doesn't mean "referenced". It doesn't mean "appears". The only way to guess what it does mean is to reverse engineer why the sentence might be here and what implementations are likely to do. Ref 02-276, 02-250r1.

W17. Wrong use of "may" in Note 16.6. Ref 02-276, 02-252r2.

W18. The sentence at [211:30-31] is a word-for-word duplicate of the one at [210:28-29], which is where it belongs.

W19. The phrase "unallocated or pointers that are not associated" at [287:10-11] is confusing, grammatically nonparallel, and different from wording used for the same thing many other places in the draft. I suggest adding "allocatables" after "unallocated". Ref 02-276, 02-259r1.

W20. The use of the term "ID=3D specifier" at [194:25] is incorrect (operations do not have specifiers), different from usage elsewhere, and directly belies the second sentence of 9.5.1.8, which says that this value is referred to as an indentifier. Ref 02-276, 02-259r1.

W21. The use of the term "rounded value" on [230:26] is incorrect; this is not a rounded value of anything. More appropriate would be "rounding value". Ref 02-276, 02-259r1.

W22. The condition on w-n being positive does not belong with> the other items in the "where" list at [231:1]. In its current context, it leaves the case where w-n is nonpositive unspecified instead of prohibited. Ref 02-276, 02-259r1.

W23. At [237:1-3] it is unclear whether the "if" condition also controls the second clause. Ref 02-276, 02-259r1.

W24. The antecedant of "its" at [257:22] is unclear. Ref 02-276, 02-259r1.

W25. At [443:12], "can" is used inappropriately where permission is intended. The placement of "only" is also questionable. Ref 02-276, 02-250r1.

W26. In Note 4.18, there is no such thing as a bound for an <explicit-shape-spec>. This probably intends "in" instead of "for". Also the sentence parses ambiguously, possibly referring to a bound for a <type-param-value>. Ref 02-276, 02-250r1.

W27. It is unclear precisely what "declared in the scoping unit of a module" means on [249:1] and [249:4]. Depending on how that is interpreted, this material might not now address the questions of the interpretations that they were an answer to. Also, the "explicitly" at [249:1] is redundant and thus misleading. Ref 02-276, 02-263r1.

TECHNICAL ISSUES

T1. I believe it to be a mistake to introduce the concept of iostat values corresponding to "harmless conditions" in the FLUSH statement. This seems inconsistent with all the other IO statements. For example, rewinding or backspacing a file that is already at the initial position might be considered a "harmless condition". Why does it have no effect to flush a file that does not exist, but causes a "harmless condition" to flush a file that is not appropriate for some other reason? This requirement is also arguably contradicted by the first sentence of 9.11.

Further the normative text describing this condition uses the adjective "processor-dependent" so much as to make it pointless and ill-defined. The "such as" does not constitute a useful definition, and the bit about inappropriate units is in a note, so all we really have is the statement that processor-defined negative values indicate processor-defined conditions.

I think this whole concept is a poorly integrated last-minute addition that stands out as anomalous. It is not necessary for or unique to the functionality of the FLUSH statement.

T2. I believe the COMPATIBLE rounding mode to be a mistake (and to be strangely named). The only difference between COMPATIBLE and NEAREST relies on an exact equality test for a floating point value. Furthermore, this exact equality test will often be a test for exact equality to a number that cannot be represented exactly (what binary floating-point number is exactly halfway between the decimal numbers 0.1 and 0.2?). This is the kind of test that programmers are incessantly warned against by textbooks and even by some compilers.

It is also manifestly unclear what it is that this mode is supposed to be compatible with. It would seem just as meaningful to name the mode RALPH.

T3. Item (2) in 1.6.3 is factually incorrect. F77 did not leave the SAVE attribute processor-defined under any circumstances. It is quite explicit about initialized variables becoming undefined if they have been modified and have not been specified in SAVE statements. The only thing processor-defined here is what the procesor will do with non-standard code that references the undefined variables. That is not appropriate for this section. And even if it is appropriate, the statement here now is factually incorrect.

T4. Item (3) is likewise about non-standard code. As the item mentions, the f77 standard said that it was not allowed to read more data than a record contained. Period. Any program that did so was therefore, non-standard. This item is thus not about standard-conforming code that now has a different interpretation. This is about code that was formerly non-standard, but is now standard. There are many, many codes that were formerly non-standard but are now standard. This includes any code that uses new f90 features. I see no reason to single out this one. If we do mention it, we need to avoid incorrect statements about it (such as that valid f77 code could have a different interpretation in f90 because of this).

T5. The terms "local entity" and "global entity" seem to be misused regularly. Introduction of the term "local name" would probably go a long way towards fixing these confusions. Paper J3/01-163 brought this issue to my attention; I think it was a good start, but didn't fix these terms.

For example, the section on host association says that if a name appears in any of a bunch of contexts, "then it is the name of a local entity". This is apparently intended to distinguish it from host-associated names, but of course, it doesn't. And the phrase "local entity of" (as used in the glossary entry for "automatic data object") makes no sense. Section 14.1.1 talks about "a name that identifies a global entity" in a way that apparently forgets that global entities may be identified by local names (via rename). (surprised there hasn't been an interp on that one because it sure sounds like it is saying that a widespread practice is illegal).

I do note that the terms "global ent" and "local ent" (I searched on these short forms to avoid missing plurals) are almost entirely restricted to c14. The only other appearances of "local ent" are two in the glossary - one is the glossary definition of the term itself; the other is the incorrect usage cited above. The only other appearance of "global ent" is one incorrect one in c12 (where it confuses the name and the entity).

I think almost all of the occurances of "local entity" would be better replaced by "local name" (which I see 6 uses of, but no definition). "Global name" also appears 6 places, none of them being in a definition. Heck, seems like many of the occurances of "local entity" already end up looking like "name that identifies a local entity". Not all the uses can be so replaced (after all, there is brief discussion of local entities that don't have names), but I think most cases can. The term "local name" seems more intuitively appropriate anyway. At least to my ear, saying that something is a local name in a scoping unit is a lot less likely to give an incorrect impression that the named entity is somehow local *ONLY* to the scoping unit in question.

T6. The section on the procedure declaration statement (12.3.2.3) has a whole bunch of syntax with no associated meaning. It adequately discusses the <proc-interface>, but that's about all.

It says not a single word about what any of the <proc-attr-specs> mean. We don't need to say much about them here, but we *DO* need to say something. As is, we don't even say that they specify the procs in question to have those attributes. We ought to at least say that, and then xref the appropriate sections. Should do the xrefs in such a manner as to also pull in all the associated constraints/restrictions. For example, nothing here says that the entity has to be a dummy argument to have the optional or intent attributes - or that the entity had better not be a dummy argument of it has the save attribute, etc.

Also, I see no hint here about what the =3D>null() is supposed to mean or what contexts it is allowed in. Perhaps that one can also be referenced. But if so, note that we also use the <proc-decl> bnf in procedure pointer components, where I think the syntax means something slightly different (default initialization vs object initialization).

T7. In the get_command, get_command_argument, and get_environment_variable intrinsics, it seems inconsistent to have a nonzero status return for character length truncation.

Such truncation is not an error or specially signaled condition in any other context, such as values returned from io specifiers. It isn't an error if you try to read a line into a character variable and the character variable is too short to hold the whole line. It isn't an error on assignment. The optional length argument does provide a way to detect this condition if the user wants to for these intrinsics.

T8. The descriptions of C_LOC and C_F_POINTER are so incomprehensible as to constitute technical, rather then mere wording flaws. The descriptions are full of requirements that are assumed rather than stated, and other requirements that contradict each other by failing to state assumed conditions. For the simplest example, [384:20] says without qualification "FPTR shall be scalar", although there is lots of discussion of cases where it is an array; this requirement is clearly intended to apply only in some cases, but that isn't stated. Ref 02-276, 02-230r3.

T9. There is no such thing as "an interface body that is not in an interface block" [78:34-35]. Ref 02-276, 02-238r2.

T10. It makes no sense to say "a dummy argument,... or a local entity" at [252:13-14], insomuch as a dummy argument is a local entity. This probably meant to say "local variable". Ref 02-276, 02-240r1.

T11. The term "subobject selector" is used in 12.4.1.6(5), but never defined. Ref 02-276, 02-241r4.

T12. The "in an actual argument list" at [20:15] is almost certainly incorrect. THis probably should be "as an actual argument". Ref 02-276, 02-241r4.

T13. The sentence at [41:20-21] directly contradicts material in 4.5.1.8. It also misuses "when" to refer to a place instead of a time. The sentence adds nothing useful and is out of place anyway. Ref 02-276, 02-242r2.

T14. The word "specific" at [55:5] rules out overriding of final bindings. I doubt this was the intent. This should probably be "nongeneric". Ref 02-276, 02-242r2.

T15. I don't understand the inclusion of abstract interface at [260:5], particularly as that inclusion was argued to be for compatability with similar words in 5.1.2.6, which were modified at the same J3 meeting to remove abstract interfaces from ths list there. Ref 02-276, 02-244r2, 02-238r2.

T16. At [45:27] the draft leaves the kind of integer unspecified in the case where the <type-param-def-stmt> omits the optional <kind-selector>. The material in 5.1.1.1 does not cover this case because there is no type specifier here. (The syntax is identical to that of a type specifier, but it is not described as one; perhaps it should be). Ref 02-276, 02-247r1.

T17. In 9.10.4 there are two references to constants, which are subject to the same misinterpretations as similar forms fixed by 02-249r1. The phrase "of the ISO_FORTRAN_ENV intrinsic module" should be added before both references to 13.8.3.2.x here. Ref 02-276, 02-249r1.

T18. Acording to [255:30-33], all abstract interface blocks are also specific interface blocks (as they lack generic> specifications). This is clearly wrong. Ref 02-276, 02-250r1.

T19. Should "END ABSTRACT INTERFACE" be allowed at [254:23]?

T20. The change of the position of the word "only" at [131:9], intended to be editorial, constitutes a major unintended technical change from previous standards. This requires the processor to do what was formerly at its option. Ref 02-276, 02-250r1.

T21. The positioning of the word only at [272:22] and [272:32] results in nonsense. This is supposed to be talking about being "only specific", but instead is now talking about "only in a scoping unit". Ref 02-276, 02-250r1.

T22. The omission of "only" from the heading of 12.4.4.2 is another technical error. It is critical that the procedure is established to be "only specific", as opposed to specific (and possibly also generic). Ref 02-276, 02-250r1.

T23. The issues involved with the output argument of defined assignment and derived type input seem similar enough that I would think the same INTENT attribute choices would be appropriate for both. Defined assignment allows either intent(out) or intent(inout) at the user's choice. Derived type input requires intent(inout). Is there reason for this apparent inconsistency? Ref 02-276, 02-269r1.

T24. The "specific or type-bound" at [397:15] is confusing because the type-bound procedures that it refers to are also specifics. The wording at [56:3] is probably misleading in this regard. Ref 02-276, 02-270.

T25. The word "contains" at [401:28] is ambiguous and confusing in this context and would probably be better replaced by "is the host of" (as opposed to the other possible reading as just "is"). Ref 02-276, 02-270.

FEATURES

F1. I propose that the enum feature be deleted.

As currently constituted, it adds almost no functionality to the language. Except for the BIND(C) case, it is nothing but another syntax for functionality that we already have. The only new functionality of the feature is the kind determination of BIND(C), which doesn't appear to be much of an issue in practice since apparently most C compilers use C int for all enums.

If a future revision ever does add a strongly typed enum feature, the overlap between that and the current one will contribute to the language seeming bloated and haphazard. I believe this potential negative impact to outweigh the small benefits of the current feature.

Adding a strongly typed enum feature now is not reasonable in my opinion. It would have too many potential interactions with oher features and would consequently cause substantial delay in the standard.

Deletion of this feature has little impact on the rest of the standard. The enum feature is currently nothing other than an alternate syntax for defining a type alias and a set of named constants; no other features depend on whether this or some other syntax for the same thing is used.

F2. I propose that an intrinsic to execute a system command be added. The CD adds intrinsics to obtain information about the command used to execute the Fortran program. The ability to in turn execute some other program is the other side of the same coin, often asked about in almost the same breath. Almost all processors have such a facility. It begs for standardization. Yes, the interpretation of the command depends on the processor. Nonetheless, this is a very useful and widely used facility. It is also trivial to add to the standard, being just an additional intrinsic with no special interactions.

This functionality is indirectly available via C interop, but I think it merits direct support, just like the intrinsics to obtain command-line information have direct support in the CD.

--
Richard Maine | Good judgment comes from experience;
maine@altair.dfrc.nasa.gov | experience comes from bad judgment.
| -- Mark Twain

References

02-007r3, Fortran 2000 Committee Draft

[End of J3 / 02-321]