Subject: Miscellaneous items
From: Van Snyder
References: 00-240, 00-317, 00-318, 01-115

Here are several things that may or may not need attention. I offer edits for a few, but mostly I don't offer edits, or I offer crappy ones.

1 Edits

Edits refer to 01-007. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by immediately following text, while a page and line number followed by + (-) indicates that immediately following text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

[Editor: Add "compatibility, Fortran 95" to the index.] 3:12

[The paragraph is about compatibility of Fortran 2000 to Fortran 95, but it's in the subclause about compatibility of Fortran 95 to Fortran 90. Editor: Move to [3:19+].] 3:36-39

[Editor: Add to the note "If the concept of 'direct component' is removed, make sure also to remove the discussion from C.1.1."] 38:32

[The note ought to apply specifically to the syntax term *data-component-def-stmt*, not generally to *component-def-stmt*. The note could be repaired, but there's no good reason to keep it: it says nothing that's not said better by the syntax rule at [40:1] and the constraint at [40:33-34], which are both on the same page; indeed, the constraint is on the previous line! Editor: Delete.] 40:35-37

[The note ought to mention type-bound procedures, at least once. Also, the train of of's is awkward. Editor: at [48:29] "independent of" ⇒ "independent from"; after "components" insert "or procedure bindings".] 48:29-32

[The sentence "A type alias ... *structure-constructor*" is misleading because it is incomplete: it doesn't include "or *parent-type-name* in an extension type definition". The sentence could be repaired, but it's just a wordy version of the previous one (because of the word "entity"). Saying it twice is an opportunity to get it wrong, or mislead by being incomplete. Editor: Delete the sentence.] 57:41-42

[The right-hand-side of R453 (for *type-alias*) refers to *declaration-type-spec*. Then there's a constraint that prohibits the CLASS keyword. If one removes from consideration the parts of R502 (for *declaration-type-spec*) disqualified by the constraint, one is left with just *type-spec*. Interestingly, "*type-spec*" is used at [58:9]. Editor: Delete "*declaration-*" twice at [58:2,6] and delete the constraint at [58:5].] 58:2,5,6

[When one gets to the end of the sentence one might reasonably ask "same as what?" Editor: "each *ac-value* expression" ⇒ "all *ac-value* expressions".] 61:9

[Unlike the note at [40:35-37] that is the subject of a nearby previous remark, this one refers to the correct syntax term, and the syntax terms and constraints from which one can derive it are up to two pages away. Nonetheless it serves only a marginal role. Editor: Delete.] 65:24-26

[The sentence "The appearance ... *entity-decl-list*" duplicates the constraint at [64:42-43]. 69:25-27

Saying something twice is an opportunity to get it wrong (twice). Editor: Delete this sentence.]

[Editor: After "associated" insert "with a target".] 96:14

[`a%kind` is not a function reference. Editor: "`Equivalent to`" ⇒ "`Same value as`".] 96:24

---

or *type-param-name* 108:7+

[The first two sentences of the paragraphs at [116:19-22] and [117:38-41] are identical, except for "specification function" on page 116 and "initialization expression" on page 117, "entity" on page 116 and "object" on page 117, and "specification inquiry" on page 116 and "inquiry function" on page 117. The first difference is to be expected; the others are not.
Editor: "entity" ⇒ "object". 116:20
Editor: "inquiry function" ⇒ "specification inquiry".] 117:41

[Editor: "*pointer-object*, the" ⇒ "*pointer-object*, the *target* shall not be a disassociated pointer and the". This is needed because a disassociated array pointer is not defined to have zero size.] 134:15

[This is just an anemic version of [75:2-4]. Again, saying something twice is an opportunity to get it wrong (twice). In this case, it's just incomplete. Editor: Delete.] 134:28-29

J3 internal note 182:20+

Unresolved issue xxx
User-defined derived-type input/output, especially the type-bound variety, is pretty pointless if it's defined in terms of list items instead of effective items. It is defined in terms of effective items in 14.1.2.4.3, but here it's a queer mixture of list items and effective items, and the definition of "effective item" is murky when viewed in this light. Defining user-defined derived-type input/output in terms of list items is comparable to the situation for intrinsic assignment: All one need do to subvert it is to define components of a type for which user-defined derived-type input/output is specified within a new type for which it is not specified. We should get this right, now, instead of trying to shoe-horn in an upwardly compatible modification in a future revision of this standard. Most of the rest of this subclause needs to be revised, and the term "effective item" needs to be defined for namelist input/output. We should also give some thought to whether a list item in an unformatted input/output list is an effective item, or (arrays of) derived type objects should be decomposed into components of intrinsic type, or of a type for which user-defined derived-type input/output is specified.

[Don't require intrinsic I/O to be a type-bound procedure with dynamic binding:] 182:33+
An effective item shall not be polymorphic unless it is processed by a user-defined derived-type input/output procedure. Same ¶

[User-defined derived-type input/output is defined in 14.1.2.4.3 in terms of effective items, but effective items are not defined for unformatted input/output.
Editor: Before "treated" insert "is an **effective item**. It is". 182:43
Editor: "list" ⇒ "effective" to make it obvious that 14.1.2.4.3 may apply.] 182:44

[Editor: Delete OPTIONAL and INTENT because they're already prohibited in BLOCK DATA subprograms by the constraint at [64:29-30], and delete PUBLIC and PRIVATE because they're already prohibited in BLOCK DATA subprograms by the constraint at [69:44].] 238:17

[Editor: After "explicit" insert ", whether it is a pointer".] 242:24

If the dummy argument is not allocatable and the actual argument is allocatable, the actual argument shall be currently allocated. 255:32+ New ¶

[The phrase "an elemental intrinsic actual procedure may be associated with a dummy argument that is not elemental" at [258:5-7] leads one to believe that dummy arguments can be elemental. Also, "dummy argument" should be "dummy procedure". Editor: "argument that is not elemental" ⇒ "procedure (which cannot be elemental)".] 258:6-7

[Editor: Delete the "have" at the end of the line. It's inconsistent and confusing to say "have" in two of four reasonable places. Better to say it just once because saying it four times would be unbearably repetititive.] 343:19

[Section 15 is more closely related to intrinsic procedures and modules (section 13) than to the material of sections 14 or 16. Editor: Move section 15 to be after section 13.] §15

2 Alphabetizing stuff without changing it

Put 5.1.2.* into alphabetical order according to the attribute name(s). Alphabetical order would be 5.1.2.9 (ALLOCATABLE), 5.1.2.12 (ASYNCHRONOUS), 5.1.2.15 (BIND), 5.1.2.4 (DIMENSION), 5.1.2.10 (EXTERNAL), 5.1.2.3 (INTENT), 5.1.2.11 (INTRINSIC), 5.1.2.6 (OPTIONAL), 5.1.2.1 (PARAMETER), 5.1.2.7 (POINTER), 5.1.2.2 (PRIVATE and PUBLIC), 5.1.2.5 (SAVE), 5.1.2.8 (TARGET), 5.1.2.14 (VALUE), 5.1.2.13 (VOLATILE). 5.1.2

Put the subsubclauses of subclause 5.2 in the same order as the subsubsubclauses of 5.1.2. Best of all, put both areas into alphabetical order according to the attribute name. Alphabetical order would be 5.2.6 (ALLOCATABLE), 5.2.10 (ASYNCHRONOUS), 5.2.13 (BIND), 5.2.14 (DATA), 5.2.5 (DIMENSION), 5.2.1 (INTENT), 5.2.2 (OPTIONAL), 5.2.9 (PARAMETER), 5.2.7 (POINTER), 5.2.3 (PRIVATE and PUBLIC), 5.2.4 (SAVE), 5.2.8 (TARGET), 5.2.12 (VALUE), 5.2.11 (VOLATILE). 5.2

Put the BNF rules for the non-optional specifiers into alphabetical order. 170:42-12

Put 9.4.4.* into alphabetical order according to the specification name. Alphabetical order is 9.4.4.3 (ACCESS), 9.4.4.8 (ACTION), 9.4.4.12 (ASYNCHRONOUS), 9.4.4.6 (BLANK), 9.4.4.11 (DECIMAL), 9.4.4.9 (DELIM), 9.4.4.1 (FILE), 9.4.4.4 (FORM), 9.4.4.10 (PAD), 9.4.4.7 (POSITION), 9.4.4.5 (RECL), 9.4.4.13 (ROUND), 9.4.4.14 (SIGN), 9.4.4.2 (STATUS). 9.4.4

Put the BNF rules for the non-optional specifiers into alphabetical order. 175:42-16

Put 9.5.1.* into alphabetical order according to the specification name. Alphabetical order is 9.5.1.5 (ADVANCE), 9.5.1.7 (ASYNCHRONOUS), 9.5.1.11 (BLANK), 9.5.1.9 (DECIMAL), 9.5.1.12 (DELIM), 9.5.1.1 (FMT), 9.5.1.8 (ID), 9.5.1.2 (NLM), 9.5.1.13 (PAD), 9.5.1.4 (POS), 9.5.1.3 (REC), 9.5.1.10 (ROUND), 9.5.1.14 (SIGN), 9.5.1.6 (SIZE). 9.5.1

Put the BNF rules for the non-optional specifiers into alphabetical order. 194:3-8

Put the BNF rules for the non-optional specifiers into alphabetical order. 197:7-40

Put 9.8.1.* into alphabetical order according to the specification name. Alphabetical order is 9.8.1.7 (ACCESS), 9.8.1.20 (ACTION), 9.8.1.27 (ASYNCHRONOUS), 9.8.1.18 (BLANK), 9.8.1.29 (DECIMAL), 9.8.1.24 (DELIM), 9.8.1.9 (DIRECT), 9.8.1.2 (EXIST), 9.8.1.1 (FILE), 9.8.1.11 (FORM), 9.8.1.12 (FORMATTED), 9.8.1.26 (ID and PENDING), 9.8.1.6 (NAME), 9.8.1.5 (NAMED), 9.8.1.4 (NUMBER), 9.8.1.15 (NEXTREC), 9.8.1.3 (OPENED), 9.8.1.25 (PAD), 9.8.1.16 (POS), 9.8.1.19 (POSITION), 9.8.1.21 (READ), 9.8.1.23 (READWRITE), 9.8.1.14 (RECL), 9.8.1.28 (ROUND), 9.8.1.8 (SEQUENTIAL), 9.8.1.30 (SIGN), 9.8.1.17 (SIZE), 9.8.1.10 (STREAM), 9.8.1.13 (UNFORMATTED), 9.8.1.22 (WRITE). 9.8.1

Put 9.9.* into alphabetical order according to the specification name. Alphabetical order is 9.9.4 (END), 9.9.5 (EOR), 9.9.3 (ERR), 9.9.2 (IOMSG), 9.9.1 (IOSTAT).	9.9

3 Non-edits or crappy edits

If they need attention, we can develop edits at the meeting, if we have time, or insert unresolved issue notes.

Page and line numbers refer to 01-007.

Either delete 1.5.2 and 1.5.3 or add 1.5.4 FORTRAN 66 compatibility.]	1.5.2-3
I thought that we did something having to do with the relation between EXTERNAL and "defined by a means other than Fortran" but I can't find it. If I recollect correctly, a procedure defined by a means other than Fortran is an external procedure.	12:34-35, 403:36-37
Should ASYNCHRONOUS and VOLATILE be in the list?	40:3-6
"Each ... shall not" would read better as "No ... shall". Much as I prefer this, **Malcolm says** the present wording is the style that is required by ISO guidlines.	40:26
These constraints entirely cripple the usability of nonkind type parameters. If we don't allow nonkind type parameters in the specification of dimensions and parameters of components, there's no reason to have them. **Malcolm says** my concerns are unfounded – type parameters are not objects. Also see edits for [108:7+] in section 1.	40:26-27, 30-32
We need an explanation, or maybe an example, of the problem described by David Moore at meeting 154, to justify the note. I remember being concerned by that problem, but I don't remember the details of his description. One example is the case of `x = f(...)` when the optimizer is clever enough to pass a reference to `x` into `f` as a hidden argument to be used for the result variable. In this case, after `f` has computed the result and put it directly into `x`, then `x` is carefully destroyed. This seems to be more of a quality-of-implementation issue than a language-design issue. In this sort of case, the assignment doesn't take place "at" the assignment statement "after" the function invocation. Instead, it takes place within the function. The hidden "result" argument should not be finalized at invocation, even though it is effectively INTENT(OUT), because it may be the same as a "real" argument. Instead, it should be finalized immediately before a value is assigned to the result variable. The finalization is effectively moved into the function. I presume that optimizers have already had to deal with `x = f(..., x, ...)`, i.e., they can't pass `x` as a hidden result argument, because assigning to it would clobber the "real" `x` argument. The `x = x` case is more insidious; maybe the standard should address it.	57:23-24
There is no prohibition against a *type-alias-name* being defined in terms of another one, so it's presumably allowed. There is therfore apparently a need to prohibit a circular definition. A constraint that a *type-name* or *type-alias-name* in a *type-spec* in a *type-alias* shall have been previously declared (perhaps within the same statement) would suffice.	4.6
I can't find a requirement, either by constraint or ordinary normative text, that a *type-name* or *type-alias-name* in a *type-spec* shall have been previously definied. Is there one? Should there be one?	5.1
The styles of the two constraints are inconsistent. I suggest the first be changed to Constraint: An entity shall not have both the EXTERNAL attribute and the INTRINSIC	64:34, 37-38

attribute.

At [81:34-36] a principle is enunciated that a (part of) an object shall not be initialized more than once, and then it is used to justify prohibiting an object of a type that has default initialization from being specified in a *data-stmt-object-list*, but such objects can (in violation of this principle) have explicit initialization in type declaration statements. For consistency, it should either be prohibited to initialize them in type declaration statements, or allowed to initialize them in DATA statements. If we must allow only one, we allowed the wrong one: How does one initialize an array of these things that has so many elements that an array constructor for it is too big to fit within the allowed statement size? On the other hand, if statement size were not limited, this wouldn't be much of an issue. 65:45-47, 81:34-36, 403:17-18

If we had a term for "type compatible and all the kind type parameters have the same value" the discussions of argument association and generic resolution would be simpler. 69:1-4

Did the specs really say "disassociated"? This would be cool, but almost certainly "disassociated" should be "undefined". 70:37

Is the description of ASYNCHRONOUS adequate? Suppose a variable V in a common block /C/ in a procedure A is not otherwise mentioned in A, a procedure B called from A initiates an asynchronous transfer causing V to become a pending input/output storage sequence affector, B and A return before the data transfer is complete, and as a consequence of A returning /C/ becomes undefined. (1) Should V have the ASYNCHRONOUS attribute? (2) Would the ASYNCHRONOUS attribute prevent /C/ from becoming undefined? /C/ becoming undefined could affect the correctness of an asynchronous write operation. Since a scoping unit does not include scoping units defined within it, the same question applies to variables accessed by host association. Notice that the proposal in 00-317 completely solves this problem. 76:27 ff

---

Do we also need to prohibit host association? 87:11

Do we also need to prohibit host association? 90:15

This one is bizarre. `b%kind` is defined even if `b` is a zero-size array. Presumably `b(10)%kind` produces the same value as `b%kind` if element 10 exists. What if it doesn't, e.g. does `b(10)%kind` exist if `b` doesn't have an element 10? Why shouldn't it? Surely, different elements of an array don't have different kind parameters! 96:26

What if the dummy argument has assumed or deferred type parameters? At [96:14-15] and [311:14-15], type parameter inquiry is only proscribed for deferred parameters of disassociated pointers or unallocated allocatables, so inquiring about assumed type parameters of dummy arguments is presumably OK. In order to do this, it seems the optional argument of NULL ought to be required. This is also the topic of interpretation request 19, at least in the context of assumed character length. 113:40

[69:28-33] and [81:4-7] are similar but not identical. The difference isn't entirely accounted for by the fact that one is in an attribute description, and the other is in a statement description. This is perhaps another illustration that saying something twice is an opportunity to get it wrong, or at least incomplete (twice). Consolidate them, removing the duplication, and move them to 7.1.7. 7.1.7 116-118

I suggest: 129:23+

Constraint: In the case of intrinsic assignment, the *variable* and *expr* shall have the same rank or the *expr* shall be a scalar.

Constraint: In the case of intrinsic assignment, the types and kind type parameters of *variable* and *expr* shall conform according to the rules in table 7.9.

Put table 7.9 here.

This paragraph's title is "Intrinsic assignment conformance rules" so we don't need "for an intrinsic assignment statement" again. The part about "rules of Table 7.9" should be a constraint (see edit proposed for [129:23+] above). Replace by "The *variable* and *expr* shall conform in shape. If *variable* is of derived type, corresponding type parameters of *variable* and *expr* shall have the same values." 130:20-22

If these changes are not accepted, at least move table 7.9 to [130:22+]. Also see 00-318. **Malcolm doesn't like** these.

Maybe we should add "or procedure references" – or maybe we should delete "or a defined assignment statement (7.5.1.6)". 134:25

There was a discussion in /data subgroup concerning whether the associate name ought to have the POINTER or ALLOCATABLE attribute if and only if the selector does. I thought the outcome was that it ought to, but those attributes are absent here. If the outcome was that they ought not to, it wouldn't hurt to have a note here explaining why not – it's not obvious. On the other hand, if it's possible for the associate name to have the POINTER or ALLOCATABLE attributes, it would be useful: It would make it easier to allocate, deallocate and pointer-assign components that have complicated antecedents. If this needs to be done, should it be done in §14? Maybe it already is done at [357:8-22]. 152:29

For consistency with array sections and FORALL, do the very minor and purely syntactic extension, that's not part of or related to anything authorized by WG5:

R830 *loop-control*	**is**	[,] *do-variable* = *loop-limits*	154:39-40
R830a *loop-limits*	**is**	*scalar-int-expr*, *scalar-int-expr* [, *scalar-int-expr*]	154:41+
	or	*scalar-int-expr*: *scalar-int-expr* [: *scalar-int-expr*]	

Is it necessary to put this requirement on list items? It would seem that we could instead put the restriction on effective items [182:35]. 182:27-29

9.5.3 could and perhaps should be incorporated into 9.9. 9.5.3

Doesn't account for user-defined derived-type input/output procedures. 184:26-27

Replacing "components ... comprise" by "effective items (9.5.2) that result from expanding" would be more precise in the case of non-namelist input/output, but doesn't work (yet) for namelist. 187:40

9.8.1.26 is the only subclause of 9.8.1 that deals with two specifiers. Couldn't they be treated with separate subclauses? This will be a valuable change if the advice in section 2 is followed. 9.8.1.26

Unresolved issue note 124 questions the precision of the wording of the requirement that "the value of a specifier in an input/output statement shall not depend on any *input-item*, *io-implied-do-variable*,...." It should also be noted that the value of the variable associated with the IOSTAT= specifier depends, at least indirectly, on these factors. 205:26-29

This paragraph is in a subclause entitled "Character editing," and therefore doesn't apply to unformatted stream output. The slash edit descriptor is specified in 10.7.2 to create a new record during formatted stream output. Lacking an explicit indication that the description of end-of-record in 9.5.3 doesn't apply to stream input, it appears that it does. Therefore there appears to be no reason to use ACHAR(10) as an end-of-record signal during stream output. If we must have a character or character sequence that is interpreted as a new-record signal, it 217:11-17

should be provided by a named constant from the ISO_FORTRAN_ENV module.	
This paragraph also needs to mention dummy procedures and procedure pointers.	245:6-8
Subclause 14.1.2.3 has nothing to do with generic procedure references. The title ought to be “Unambiguous generic procedure definitions” or “Restrictions on generic procedure definitions such that references are unambiguous.”	343:11
The sentence “If a generic...” conflicts with, or at least belongs in [348:30-31].	343:47-2
14.1.2.4.1 takes no account of procedure pointers in generics.	345
These paragraphs do not explicitly apply to defined operations or defined assignment. They apply indirectly to defined operations by way of the phrase “it is generic in exact analogy to generic procedure names” at [248:15], but there is no parallel statement concerning defined assignment. 01-115 addresses this.	345:4-14
Either “procedure” should be “interface” at [345:8], or vice-versa at [345:14]. 01-115 does this.	345:8, 14