Subject: Miscellaneous observations concerning 00-007
From: Van Snyder

1 Introduction

The page and line numbers in the margin of this report refer to 00-007.

The principle I used in examining the standard was to be as critical as possible. Remarks that are incorrect could be ignored, but if I neglect to mention something that I think is minor, but that isn't, there's no guarantee somebody else will notice it.

I didn't have time to look other than superficially at sections 13-16 or the annexes.

If any of these observations are incorrect or ill-founded, or any of the recommendations are incorrect, tell me so I don't continue to update them in reports for subsequent meetings. I hope to be convinced that many of these observations do not need attention. Maybe the editor could make J3 internal notes of some of the remarks that have substance.

There are observations of the type "A duplicates B; should it be deleted?" and "Add A here" even though it would duplicate B. The dueling sentiments to delete and add things may seem inconsistent. Both kinds are present because I don't want to leave a stone unturned.

If an individual or subgroup agrees that some of these remarks address real problems, but nobody else has time to work on them, let me know and I'll try to write something more formal. I don't want to write something formal if it would go nowhere.

Minor specific edits, that I think ought not to be controversial, are offered in section 3.

2 Comments with questionable edits or without specific edits

The preference to use syntax terms instead of descriptive names begs for a way to get their definitions into the index. I don't think it's necessary to index every appearance in every syntax rule – indexing the left-hand-sides would be enough.	everywhere
There is no normative definition of entity.	somewhere
Section 4 is not limited as described here: There is some material in 4.2 about specifying the type parameters of objects.	xv:11-12
A brief discussion of features in the various sections is not out-of-place in the **Organization** part of the **Introduction**. Is the objection to their being identified as new in Fortran 2000? If so, the "new in Fortran 95" phrase at [xv:34] is (was) also inappropriate.	xv:17-22
Was the behavior of SIGN for negative real zero changed between FORTRAN 77 and Fortran 90? I thought it was later.	4:17-19
This looks at first like two citations for ISO/IEC-646:1991. Combine into one paragraph, and replace "ISO/IEC-646:1991 (International reference Version)" by "This". Otherwise, at least either capitalize "reference" or don't capitalize "Version."	7:3-5
This definition confuses rather than clarifies the definition of **host scoping unit** at [12:7]. Replace "called the **host**" by "is the host scoping unit".	12:39
This definition confuses rather than clarifies the definition of **host scoping unit** at [12:7]. Replace "called the **host**" by "is the host scoping unit".	12:42

The phrase "within the scoping units of the host" is confusing and incorrect. An internal procedure might be accessible within a derived type definition, but that's kind of useless. An internal procedure is not available within an interface body. Replace "scoping units of the host" by "host scoping unit." 13:2

Add "for derived-type input/output or" after "invoked." 13:5

Delete "All statements ... module." It is too arcane for the superficial level of section 2, and duplicates material in sections 11.3 and 11.4. 13:28-30

What is a "subclause?" Replace "of subclause" by "in". 13:32

Do we want to call end construct statements, e.g. `end if`, "END statements" 2.3.3

Should exponent and fraction be called subobjects? 2.4.3.1

Replace "or redefined" by ", redefined, or undefined." 16:41

Replace "and rank" by "rank, and attributes" (to account for ALLOCATABLE or POINTER attributes of the result). 17:18

Replace "and an allocatable array" by "an allocatable array, and an array that is a structure component if any of its bounds are declared by using nonkind type parameters." 17:38

Replace "two" by "four" if the change suggested for [18:37+] below is accepted. 18:28

Add new paragraphs after note 2.6: 18:37+

"A **type parameter keyword** may be used in a derived type specifier (4.5.5) to indicate the type parameter for which a value is specified.

"A **component name keyword** may be used in a structure constructor (4.5.6) to indicate the component for which a value is specified."

Note 2.6$\frac{1}{2}$

Type parameter keywords and component name keywords can make structure constructors more readable and allow type parameters or structure components to be specified in any order.

Are RECURSIVE, PURE and ELEMENTAL attributes? Maybe "or attributes" should be "attributes or other properties." 19:2

Should invokation of a procedure by derived-type input/output be in the list? 19:15-16

Add ", modules" after "procedures." 19:29

Add a new sentence: "Intrinsic modules may be accessed by use association." 19:31

The statement "A lower-case letter is equivalent to the corresponding upper-case letter in program units except in a character context" duels with exceptions for keywords in input/output statements. Those should be mentioned here, too. 23:25-26

This paragraph is self-contradictory. Something like the following would be more self-consistent: 27:12-15

"In **free source form** there are no restrictions on where a statement (or portion of a statement) may appear within a line. A line may contain zero characters. If a line consists entirely of characters of default kind (4.4.4), it may contain at most 132 characters. If a line contains a character that is not of default kind, the maximum number of characters allowed on the line is processor dependent."

Should we add ", and not greater than 132" at the end?

Should TYPE ALIAS be in the list? 28:3-10

Belongs at [30:6+]. 28:13-18

Should "in character position 6" be "before character position 7?"	29:29-30
Suppose I have a file A that consists of two lines, say `call s` and `INCLUDE 'B'`. Is it OK if the file B consists of `call s`? Doesn't this "result in inclusion of the same source text?"	30:19-21
"An intrinsic type is always accessible" duplicates 2.5.7. Is it needed here, too?	31:13-14
"A user ... parameter" would be clearer as "A type parameter of a derived type may be specified". As it stands, it suggests that a user can define a kind parameter for an intrinsic type.	32:23
Is the use of the extra digits processor dependent, or should some kind of rounding be specified here?	37:5-6
Apostrophes and quotes are not listed as delimiters in 3.2.5. "delimiter" should be "apostrophe or quote," or [39:11-12] should be moved to [38:34+].	38:35
Belongs at [42:20+] (it has nothing to do with R426).	42:23-24
Simplification: Add "or ALLOCATABLE" after "POINTER" at [43:9] and delete [43:13-14].	43:9,13-14
Now that we have nonkind type parameters, "automatic" array components seem not to be an added complication. It would seem that inquiry functions concerning previously declared components of the same type, e.g. UBOUND, were never a problem.	43:11-12
Simplification: Replace "*binding-private-stmt*" by "PRIVATE" at [44:3] and delete R438 at [44:6]. The syntax term isn't used anywhere else.	44:3,6
We probably need a constraint that all of the nonkind parameters of the passed-object dummy argument shall be assumed.	44:23+
An internal procedure wouldn't cause any problems if the type is declared in the host scoping unit of the internal procedure. If a type is declared within a procedure, can a binding be to the procedure within which the type is declared, so long as its interface is explicit?	44:27-28
The word "itself" doesn't contribute anything.	45:11
Is it necessary to specify that a type parameter that implicitly has the KIND attribute shall not explicitly have the NONKIND attribute?	46:18+
Add "! Uses type definition from Note 4.23."	47:5
A section **4.5.1.4$\frac{1}{2}$ Allocatable components** would be an obvious place to look for properties of allocatable components. E.g. they come into existence deallocated, and they get deallocated whenever the object of which they are a component "ceases to exist."	48:42+
The wording of these two conditions under which the structure constructor can't be used should be parallel. At [49:34], replace "shall be ... module" by "is accessible only within the module containing the definition."	49:28,34
Delete "that is" to be parallel in construction to [50:4].	50:2
Add "for which user-defined derived-type input/output (9.5.4.4.3) is not specified."	51:24
Should "SEQUENCE property" be "SEQUENCE property or BIND(C) attribute?"	51:38
Is the term "intrinsic input/output" defined, or should this say "formatted input/output that is not user-defined derived-type input/output (9.5.4.4.3)?" Also, the order of components is significant for namelist input.	53:13
Is it necessary to add "If a binding is deferred and the interface that would be inherited if no abstract interface were specified specifies PASS_OBJ, an abstract interface shall be specified?"	54:25+

Maybe not, since the deferred binding can't be invoked.

The constraint should be with R413 at [33:20]. 55:19-20

Needs to specify the type parameters of the constructed value. See related remarks at [115:11-12] below. 4.5.6

Parameterized *type-alias-names* would be useful, E.g. `TYPEALIAS :: SQUARE_MATRIX(K,I) => REAL(K), DIMENSION(I,I)`. An object of type `SQUARE_MATRIX` would be used like any rank-2 real object, e.g. 4.6

```
TYPE(SQUARE_MATRIX(KIND(0.0),:)), ALLOCATABLE :: M
ALLOCATE ( M(3) )
M(1,2) = 3.4
```

Whether a parameter is a kind or nonkind parameter would be inferred from its use. In the above example, K is a kind parameter, and I is a nonkind parameter. A parameterized *type-alias-name* could be used like "partial evaluation" in functional languages. E.g. `TYPEALIAS :: A(I) => B(I,3)`

How about making the ALIAS part optional? If it's absent, a new type is defined; if it's present, a new name for an existing type is defined. E.g. `TYPE :: SQUARE_MATRIX(I) => REAL, DIMENSION(I,I)` would define a new type `SQUARE_MATRIX` that could be used for generic resolution.

Much as I wish they were, enumerations aren't types. Should the text be repaired so it doesn't say they are types, or should we re-open the discussion of whether they are types? 58:15, 16, 24

What is a "corresponding enumeration type of the companion processor?" There's no section "Interoperation with C enumeration types" in 16.2. 58:16

The second sentence of this paragraph should be a constraint. 60:30-32

Simplification: Add "ALLOCATABLE," before "TARGET" at [64:24] and delete the constraint at [64:38-39]. 64:24,38-39

Constraint: The INTENT and VALUE attributes shall not be specified for dummy procedures. 64:31+

If there's a problem with assumed or deferred character length, then there's a problem with assumed or deferred type parameters in general. 65:12-27

Does this constraint serve a technical or an aesthetic purpose? If the latter, delete it. The VALUE attribute would be useful for ordinary Fortran procedures, especially if the semantics are changed slightly so that it indicates that the actual argument associated with the dummy argument won't be changed, but doesn't prevent the dummy argument from being changed. I.e., it means something like copy-in, not INTENT(IN). (BTW, C semantics are like this.) It this case, it would be better as INTENT(VALUE). Then the INTENT parts of the constraint at [65:10] would also be unnecessary. Also see remarks for 5.1.2.14 below. 65:28-29

I don't understand the need for the constraint for which unresolved issue 90 expresses a desire. 65:33-38

Is this really intended? Maybe it should be "A *language-binding-spec* shall not appear in the *module-subprogram-part* of a module." Even that is questionable if the *language-binding-spec* is specified for an external procedure. 65:42

What does the complete sentence "The entity declared shall be a variable" mean? Is this a separate constraint, or is it conditioned on "If a *language-binding-spec*...?" 65:44

Simplification: Replace "*scalar-int-literal-constant*" by "*digit-string*" at [68:1] and delete the constraint at [68:4]. 68:1,4

Is SEQUENCE a property or an attribute?	69:17
Is it necessary to remark that an unlimited polymorphic object is a polymorphic object?	69:35
"Type compatible" isn't in the index. If this is the definition of "type compatible" it needs to be bold and in the index. This is probably a poor place for the definition of "type compatible" since it applies to nonpolymorphic objects as well.	69:38
Add SELECT TYPE and ASSOCIATE to the list.	70:18
"The appearance of a PARAMETER..." is already covered by a constraint at [64:43-44].	70:27-29
Also needed at [117:10], [118:34], and [119:31].	70:30-35
The term "dereferenced" is nowhere defined. Perhaps "it is dereferenced to indicate" should be replaced by "indicates".	72:12
Doesn't cover components that are arrays. Needs "or *component-decl*" after "*entity-decl*," "or *component-array-spec*" after "*array-spec*," and "or *data-component-def-stmt*" after "type declaration statement" in several places. Maybe parallel paragraphs would be clearer: Put "The **DIMENSION attribute** specifies entities that are arrays" in a single paragraph at [72:44]. At [73:16+] add a new paragraph: "The rank or bounds of a structure component are specified by the *component-array-spec*, if there is one, in the *component-decl*, or by the *component-array-spec* in the DIMENSION *component-attr-spec* otherwise. A *component-array-spec* in a *component–decl* specifies either the rank or rank and bounds for a single array and overrides the *component-array-spec* in the DIMENSION *component-attr-spec*. To declare an array in a *data-component-def-stmt*, either the DIMENSION *component-attr-spec* shall appear, or a *component-array-spec* shall appear in the *component-decl*. The appearance of a *component-array-spec* in a *component-decl* specifies the DIMENSION attribute for the component."	5.1.2.4
The phrase "of a procedure" doesn't add anything.	73:24-25
Does construct association determine the bounds of the *associate-name* if the *selector* has deferred shape? (Also see 00-105.)	74:9
Delete "an array with ... TARGET statement (5.2.8)" because it's covered by a constraint at [64:20-21]. At least delete "in a type ... TARGET statement (5.2.8)." It wasn't found to be necessary in the next paragraph, and it's covered at 5.1.2.4 (well, almost – see remarks about 5.1.2.4 above). Also see remark at [74:20-21] below.	74:11-14
Delete "an array with ... *deferred-shape-spec-list*" because it's covered by a constraint at [64:20-21]. Also see remark at [74:11-14] above.	74:20-21
Constraint: An array declared as an assumed-size array shall be a dummy argument.	74:43+
Delete the phrase ", and shall be specified only for dummy arguments." It is covered by a constraint at [64:30-31]. The next sentence also makes it clear the OPTIONAL attribute applies only to dummy arguments.	75:42
The sentence "If the pointer is an array ..." is covered by a constraint at [64:20-21]. Is it needed here?	76:4-5
Would it help to add "Otherwise its deferred type parameters are undefined?"	76:7
The sentence "If it is an array..." is covered by a constraint at [64:20-21]. Is it needed here? Many constraints from pages 64-65 are repeated in previous sections. Should we add here that it is prohibited for an entity to have both the EXTERNAL and INTRINSIC attributes, even	76:30-31

though it is covered by a constraint at [64:35]?

Add "in the same scoping unit?"	77:36
Given that Fortran has a VOLATILE attribute (described in the next paragraph) it is confusing to refer to "the VOLATILE attribute provided by some processors." It also makes one wonder why we need both attributes. Delete "similar ... It is".	77:44-45
This seems to cover the need for the ASYNCHRONOUS attribute. Do we need it if we have VOLATILE?	78:3-4
If the ALLOCATABLE and VOLATILE attributes are both specified, does the volatility apply both to the target and the allocation status?	78:16-17
It would provide more useful functionality if the VALUE attribute were INTENT(VALUE) instead, indicating that the actual argument cannot be changed by way of the dummy argument, but that the dummy argument can be changed within the procedure. This is somewhat different from and more useful than implying INTENT(IN). Also see remarks at [65:28-29] above.	5.1.2.14
There's nothing here about what NAME= and BINDNAME= do. Surely such specification, or at least a reference to it, belongs here!	5.1.2.15
This sentence seems to say nothing about the VALUE attribute.	78:19-20
The sentence "The combination ... is subject to the same restrictions..." applies only to the combination of attributes. There are other restrictions, e.g. OPTIONAL can only be used for dummy arguments, that have nothing to do with other attributes. Replace "is" by "and the constraints on their applicability are."	79:18-19
Would be better as a constraint on the INTENT attribute, on page 64. See remarks at [64:31+] above.	79:24
Duplicates the constraint at [64:32-34].	80:25-27
This seems to prohibit a SAVE statement for a common block in the scoping unit of the main program. I think what's meant is that it's not *required* there. Add "it is not required" after "except."	80:37
Duplicates the constraint at [64:20-21].	81:16-17
Duplicates the constraint at [64:20-21].	81:26-27
Duplicates the constraint at [64:28-29].	81:28
Duplicates a small part of the constraint at [64:26-27]. Do all of it or none of it.	81:37
Duplicates the constraint at [65:42].	83:1-2
Duplicates the constraint at [65:46-47].	83:3-4
The sentence "If ... *data-stmt-object-list*" should be a constraint. Add it (with appropriate rewording) to the constraint at [83:30-36].	83:9-11
A section subscript isn't an expression, so it can't be an initialization expression.	83:28
I don't understand the need for a prohibition against the BIND(C) attribute. (1) I thought assignment semantics were used for DATA statements, in which case assignment ought to be prohibited, too. (2) There's no similar constraint for *initialization* in a *type-declaration-spec*.	83:34,35
How can a derived type be inaccessible in the local host if it's accessible in the host scope?	86:5-7
Add "or groups" after "group."	87:33

There's no explicit specification that the namelist group objects are members of the namelist group denoted by the preceeding namelist group name. Add "Each namelist group object is a member of the group identified by the namelist group name that precedes it in the NAMELIST statement."	88:0+
There is now a BIND statement – in 5.2.13. It's called a BIND statement, not a BIND(C) statement.	88:15,18-22
The constraint at [88:35-41] catches all cases but this one. Add this one to that constraint, and delete this one.	89:13-14
There is no "equivalence association" mentioned in 14.6 (at least not as a section heading). Perhaps "Storage association by equivalence" is a better (but maybe not ideal) title.	89:33
Duplicates the constraint at [64:20-21]. If the requirement at [79:18-21] that the restrictions about attributes apply equally whether they're specified by attributes of type declaration statements, or by attribute statements to [63:6+], and include COMMON in the list, this wouldn't be needed.	91:9-10
Is it OK for a *variable-name* to be accessed by host association?	91:11
See remarks at [91:9-10].	91:23-24
There is no section heading in 14.6 containing "common association." Perhaps "Storage association by common" is a better (but maybe not ideal) title.	92:1
Add "has been declared in the main program unit or its" after "block."	92:37
The phrase "according to the rules for equivalence objects (5.5.1)" is not directly supported by anything in 5.5.2.3. The rules for storage association by common and storage association by equivalence are parallel, but not explicitly identical. It would help to have a note that says they're similar (except for the TARGET attribute restriction for equivalence objects).	92:31-32
It seems not to be prohibited here to bring an object accessed by host association into a common block by way of equivalence. There's an interp pending (resolved at meeting 151?) concerning whether any attributes of an entity accessed by host association can be changed. Would that cover it? Probably not, since "in a common block" isn't an attribute.	5.5.2.5
"The appearance ... is termed a reference..." sounds like a definition. Set the word "reference" in bold face and add it to the index.	95:2
There should also be a sentence "A reference to an allocatable variable is permitted only if it is allocated and defined."	95:3-4
The note would be clarified if "even if a component is an array" were added at the end.	96:3
In a quick reading, one might conclude conclude that the rightmost *part-ref* can't be of derived type. Is this the intent?	96:36
The term "dereferenced" is nowhere defined. Perhaps "where it is dereferenced to refer" should be replaced by "that refers".	97:9
I don't think this suggestion works for allocatable scalars.	97:23-25
The term "data object" excludes functions, function procedure pointers and dummy functions. Does it make sense to inquire about nondeferred parameters of the result types of these entities? If we allow assumed parameters for the result type of a dummy function procedure pointer or dummy function (see remark at 12.3.2.3 below), it would at least be useful to inquire about them. I can't find a definition for the term "inquire about." Is this intended to work by analogy with	98:2

"inquiry function?"

This sentence should be taken out and shot. If "The appearance of a whole array in a nonexecutable statement other than in an equivalence set (5.5.1.3) in an equivalence statement specifies the entire array" says the same thing, I think it's better.	98:34-25
How about case (11) in 14.7.7?	101:26
The clause "pointer assignment (7.5.2) associates pointers with existing targets" is inaccurate (because the *target* might be NULL()). It would be more accurate to say "an existing target may be associated with a pointer by a pointer assignment statement (7.5.2)" or "pointer assignment (7.5.2) disassociates pointers or associates pointers with existing targets."	102:4-5
Add "; otherwise an error condition occurs."	103:26
Discussion would flow better if this were after 6.3.1.2.	6.3.1.1
"and is definable" seems not to add anything.	103:41
The sentences "Allocating a currently allocated..." and "The ALLOCATED intrinsic..." are concerned with allocation status, and as such should be in 6.3.1.2.	103:41-45
Add "and type parameters" after "array bounds."	105:44
"it can still be refered to by other pointers that" could be simplified to "other pointers" without losing anything.	105:47
The essence of the sentence "An allocatable variable with the TARGET attribute..." is repeated more clearly at [108:1-2]. Is it needed here, too?	107:3-4
Add "; if attempted an error condition occurs" or maybe replace the entire sentence by "If a DEALLOCATE statement containing a pointer that is currently associated with an allocatable entity is executed an error condition occurs."	108:1-2
Do these two events cause error conditions? If they do, we should say so, so that STAT= and ERRMSG= get set.	108:3-5
Replace "is determined by the constructor name" by "and type parameters are as described in 4.5.6" (but only if the repairs suggested for that section are done).	115:11-12
Shouldn't "entity" be "variable" here?	115:30
Add "intrinsic" before "operation" because the section doesn't address defined operations.	116:18
Would be clearer as "The result of an intrinsic operation has a kind type parameter. The result of an intrinsic character operation also...".	116:23
Need a section **Data type, type parameters, and shape of the result of a defined operation**. All the necessary stuff is hiding in 7.1.4.1 under the guise of function references.	116:45+
The term "restricted expression" seems not to be needed. It appears to be referenced only from the previous line and within itself. Can't we use "specification expression"?	117:9
Since a section subscript isn't an expression, it's hard to imagine how it could be a restricted expression.	117:33
Since a section subscript isn't an expression, it's hard to imagine how it could be a constant expression.	119:8
Why is the set of intrinsic functions that can be used for initialization expressions more restricted than the set of intrinsic functions that can be used for constant expressions? (Compare to [118:40-43].)	119:37-47

I'm curious why "properties" instead of the more precise "type parameters and bounds" is used here.	120:17
Since a section subscript isn't an expression, it's hard to imagine how it could be an initialization expression.	120:25
After "defined" add a new sentence "If the operand is allocatable it shall be allocated and defined."	121:19
Shouldn't this be "kind type parameters?" Otherwise assumed type parameters are prohibited.	129:38
Shouldn't this be "kind type parameters?" Otherwise assumed type parameters are prohibited.	130:9
Says the same thing as at [132:39-2], but not as completely or as well.	132:32-34
Are the normative text and note inconsistent, or does the note imply construction of a temp?	134:29-35
Shouldn't this be "kind type parameters?" Otherwise assumed type parameters are prohibited.	135:4
I wrote a note to myself "It should be explicitly spelled out what happens if *variable* and *expr* overlap." I think this was intended to apply to assignment in general, not just to defined assignment.	135:43+
One should be able to learn here that using NULL() for *target* results in nullifying the *pointer-object*, but one can't except indirectly in the case of assignment to a procedure pointer.	7.5.2
(1) I didn't know that expressions "delivered" anything. Replace by "The result of *expr* shall have the POINTER attribute." (2) After the pointer assignment takes place, does the pointer result of the *target* get deallocated? Pointer results of functions get deallocated "after use." Should there be an exception for the case when a function with a pointer result is used as the *target* in a pointer assignment?	136:45
Should be a constraint.	140:43
Add "other than restoring the control mask and pending control mask of an enclosing WHERE construct."	141:2
Replace "usually" by "may be." Asserting "usually" implies some foreknowledge of the program. (At least the "usually" shouldn't be normative.)	148:9
This constraint could be done with syntax rules.	149:27-28
The phrase "and execution continues as though a CONTINUE statement (8.3) were executed" contributes nothing, since a CONTINUE statement does nothing.	149:31
(This is the same area at which 00-105 proposes changes. This should maybe be in 00-105.) Add "or is a *variable* that has a vector subscript" after *variable*. Add "within the SELECT TYPE construct" at the end of the constraint. Add another constraint: Constraint: If the *selector* is a variable that is a dummy argument with the INTENT(IN) attribute, *associate-name* shall not appear in a variable definition context (14.7.7) within the SELECT TYPE construct.	152:29-30
Add "within the ASSOCIATE construct" at the end of the constraint. Add another constraint: Constraint: If the *selector* is a variable that is a dummy argument with the INTENT(IN) attribute, *associate-name* shall not appear in a variable definition context (14.7.7) within the ASSOCIATE construct. (00-105 does this.)	154:36
The rules concerning attributes of the *associate-name* should be the same for SELECT TYPE and ASSOCIATE constructs. If the ASYNCHRONOUS, VOLATILE and INTENT attributes	155:2-8

of the *selector* apply to the *associate-name* (at least when the *selector* is a variable), then the POINTER and ALLOCATABLE attributes, and pointer association or allocation status, should apply as well. Then, it wouldn't be necessary for the selector to be associated or allocated, and the two constraints above about INTENT(IN) wouldn't be needed. (00-105 does this.)

This constraint could be done with syntax rules.	156:39
Long ago, in a galaxy far, far away, the *do-term-action-stmt* couldn't be a logical IF statement if its consequent couldn't be a *do-term-action-stmt* on its own. Has this requirement intentionally vanished?	156:39
Same two remarks as for [156:39] above.	157:7
Given that we now have the concept of ERROR_UNIT, it would be better to issue the warning on it than on the unit identified by *. Change "* in a WRITE statement" to "the named constant ERROR_UNIT from the ISO_FORTRAN_ENV intrinsic module (13.17.1.3)".	161:20
Some discussion in section 9 refers to statements by their categories defined in this paragraph. In what category is the WAIT statement?	163:12-15
Add "processor-dependent" before "restriction".	164:31
Add ", assuming a READ statement for this connection is allowed" (compare to [166:27-28]).	166:10
Does the "position just after the last record" mean that it's just after the last data record, or just after the endfile record? (See [164:17].)	167:19, 25, 31
Does the "otherwise" refer to direct access, stream access, or output?	168:2
Does the "otherwise" refer to direct access, stream access, or input?	168:8
The phrase "that is not..." duplicates the constraint on R903, and as such is not needed.	169:41-42
The syntax rules already say this. It's not necessary to say it with text. If it is necessary, at least add the WAIT statement here, and at [170:25].	170:41-42
"of default character type" is said thrice already, once in a constraint. Is it necessary to say it again?	172:7-8
Are the ERR= and IOSTAT= specifiers "in effect"? I am confused by this sentence because I think they're not.	172:31
Belongs in 9.4.4.2.	172:41-44
Belongs in 9.4.4.2.	173:3-4
Add something to require that the branch target could be accessed by a GO TO statement from the point of the OPEN statement.	173:27-28
Should perhaps be in 9.4.4.1.	173:29-31
The sentence "The *file-name* shall be a name that is allowed by the processor" is repeated at [200:18-19], but more precisely. Is it needed here, too? The sentence "If this specifier ... processor-dependent file" and the material at [173:29-31] should be together.	173:43-46
Would be clearer if "... the endfile record is the next record, if it has one" were "... the endfile record, if it has one, is the next record." (Upon first reading, I thought "one" referred to "the next record," not "the endfile record.")	175:5-6
Belongs in 10.7.7.	176:17-177:26
Rounding needs to be defined in terms of the external (decimal) representation. I don't think	176:29-33

anything else can work.

The *external-file-unit* isn't optional in the CLOSE statement. What does "that refers to a unit" mean? Remove it, and replace "that unit" at [177:31] by "the unit specified in the CLOSE statement."	177:29, 31
"with status ... DELETE" duplicates 9.4.5.1. Delete it, and "Note 9.20 The effect is" (making the rest of the note normative).	177:41-44
Add something to require that the branch target could be accessed by a GO TO statement from the point of the CLOSE statement.	173:27-28
The "exactly one" constraint is done differently for data transfer statements, as compared to the OPEN statement (see [173:26]). The constraint at [179:26-27] implies at least one. Replace this constraint by Constraint: Each specifier shall not appear more than once in a given data transfer statement.	179:17-18
Add something to require that the branch target could be accessed by a GO TO statement from the point of the data transfer statement.	179:20-22
What happens during namelist input?	185:29
Is it possible to define the variable specified in a SIZE= specifier if an error occurs?	186:27
Is it possible to define the variable specified in a SIZE= specifier if an error occurs?	187:5
This sentence seems to have little or no relation to establishing the direction of data transfer. It should be step (1.5) or (2.5) in 9.5.4.0.	187:32-33
Is it necessary to repeat "If the format..." here? It's already at [180:38-39] – in the definition of the format specifier.	188:5-6
Appears to be inconsistent with the requirement for a REC= specifier in data transfer statements that refer to units connected for direct access. (See [166:3-4].)	190:8-9
Insert **NOTE 9.42$\frac{1}{2}$** INTENT(INOUT) is used for the `errmsg` variable because it is not changed if no error occurs. The initial value is not used.	192:13+
Would it be better not to have an intent for the `errmsg` variable?	*Question*
Need something like the previous paragraph for `TYPE(whateveritis)`.	192:15+
Categories of input/output statements were earlier used in similar situations. Should WAIT be in this list?	194:13-14
The statement "A child data transfer statement that is a list directed or namelist input/output statement may contain a list item of derived type" makes one wonder if other data transfer input/output statements also can contain list items of derived type. If it's correct to say "Any child data transfer statement, including a list-directed..." then perhaps that should be said. If it's correct to say "Only a child data transfer statement that is a list-directed..." then perhaps that should be said.	194:17
"Usually" prejudges program style. At least "usually" shouldn't be normative. I would use "could be" or "may be."	194:22
Given that there is a complete list, two lines below, of edit descriptors that have state, is there any point to keeping the partial list "such as BN, BZ, SP, and P" here?	194:30
The title "Waiting on pending data transfer" sounds like it's about to get coffee brought to it.	195:31

Change "on" to "for."

The note is the same as the last sentence of the introductory paragraph of the section, which is only four lines earlier. Is there any point to keep it?	196:5
The constraint at [196:16-17] implies at least one unit specifier. It's shorter to do this like for the OPEN statement: Constraint: Each specifier shall not appear more than once in a given *wait-stmt*.	196:14-15
Add something to require that the branch target could be accessed by a GO TO statement from the point of the WAIT statement.	196:18-20
Isn't it clearer to say "specified by" instead of "in accordance with."	197:9
Add something to require that the branch target could be accessed by a GO TO statement from the point of the file positioning statement.	197:27-28
The constraint at [197:29-30] implies at least one unit specifier. It's shorter to do this like for the OPEN statement: Constraint: Each specifier shall not appear more than once in a given *position-spec-list*.	197:31-32
It is only for INQUIRE that we say "All specifier value assignments are performed according to the rules for assignment statements," and not for any other input/output statements. Either it's needed for the others, or not needed for INQUIRE. Since the description of each specifier that can be assigned a value says "assigned" I think the sentence isn't needed here. If it is needed, it should apply to all input/output statements.	198:39-40
Constraint: The *label* in the ERR= specifier shall be the statement label of a branch target that appears in the same scoping unit as the INQUIRE statement. Add something to require that the branch target could be accessed by a GO TO statement from the point of the INQUIRE statement.	200:7+
The first two sentences of 9.8.2 are covered by a constraint at [200:2-3], so they're not needed here. Delete them. The remaining sentence is about the unit specifier, so change the title to "UNIT= specifier in the INQUIRE statement."	205:1-6
Are any of the problems in unresolved issue 3 [186:10-18] reduced by writing "The input/output operation terminates?" Similarly, are any of those problems reduced by writing "The input transfer terminates" at [206:26, 207:7]?	206:24
Is it possible to define the variable specified in a SIZE= specifier if an error occurs?	206:30
The term "internal unit" appears to have been used nowhere but here. The other places all use "internal file."	207:24
It's probably necessary to specify which namelist group is constrained.	207:28-29
Simplification: Replace "*int-literal-constant*" by "*digit-string*" at [210:26] and delete the constraint at [210:30].	210:26, 30
Simplification: Replace "*int-literal-constant*" by "*digit-string*" at [211:3-7] and delete the constraint at [211:11].	211:3-7, 11
Simplification: Replace "*int-literal-constant*" by "*digit-string*" and delete the constraint.	211:23-24
Simplification: Replace "*int-literal-constant*" by "*digit-string*" at [211:29] and delete the constraint at [211:31].	211:29, 31
Move the paragraph to [213:14+], and include an exception for reversion (which is discussed at [213:3-14]).	212:27-28

"input/output" should be "effective" (so as to apply to components of derived type objects).	215:19
Replace "complex" by "the real or imaginary part of a complex object" because the sentence is talking about *one* edit descriptor.	215:21
Should the reader be alerted about the effect of SP edit descriptors on optional plus signs?	215:43
Add "as specified by the established rounding mode (10.7.7)" after "digits."	216:1
There are several dots in the template. Replace the dot by "The . before x_1."	216:15
Add "as specified by the established rounding mode (10.7.7)" after "rounding."	216:16
Add a note similar to note 10.11 and 10.12 (in the next two sections).	216:33+
There are several dots in the template. Replace the dot by "The . before x_1."	217:8, 39
It's unfortunate – depending upon one's view of irregularity – that the *d* in a G edit descriptor isn't used for integer editing. After all, it *is* used if it's in an I edit descriptor.	219:35
Should RC and RP be in the list?	223:44
Much of the discussion of list-directed and namelist formatting would be clearer and simpler if syntax rules were developed for them.	10.9, 10.10
Replace "commas" by "nonblank value separators."	226:20
Replace "characters blank, comma" by "value separators blank, comma, semicolon".	226:28
Replace "blank, comma, slash" by "value separators".	226:36
Insert "nonblank" before "value" twice.	227:1-2
Replace "slash or comma" by nonblank value separator".	227:22
Insert ", a preconnected file that has not been opened" after "file".	228:20
Replace "The" by "A namelist comment or the".	229:5
I could find no precise specification that the namelist group used for namelist formatting is identified by the *namelist-group-name* in the data transfer statement. There is only a hint given by the syntax term. Add "in the namelist group identified by the *namelist-group-name* in the data transfer statement" after "(5.4)".	229:8
Replace "NAMELIST" by "data transfer".	229:23
I couldn't find a definition of "qualify".	229:31-1
Insert "no scale factor and" before "no".	230:25
Replace "comma" by "nonblank value separator" four times.	231:2-8
Replace "part" by "field".	231:4
Insert ", semicolon" after "comma".	231:17
Insert "nonblank" before "value" twice.	231:34-35
Replace "slash ... and" by "nonblank value separator, and".	232:3-4
Since this paragraph describes how namelist data begins, move it to [233:33+].	233:39-42
Now that we have program arguments with sizes not given by initialization expressions, it would be useful to allow automatic objects in the main program. The only things they could depend on would be lengths and bounds of program arguments, but this could be useful.	235:22-23
Could be specified as constraints, and would be stronger if done so.	237:2-4

Replace "called the **host**" by "is the host scoping unit."	237:16
Should "procedures and types" be "entities", in case an intrinsic module ever exports a parameter or .... (Yes, I know the only intrinsic things we've defined so far are procedures and types.)	237:24
Are the empty parentheses intended to contain cross references to R703 and R723?	239:11
"ONLY qualifier" was "ONLY option" just a few lines above. Which term should be used?	239:20-21
"[a] USE statement" should be "use association".	239:35-36
should "made accessible by a USE statement" be "accessible by use association"?	239:42
"ONLY clause" was "ONLY option" and then "ONLY qualifier" on the previous page. Which term should be used?	240:17
Constants are objects, and named constants get their values with an *initialization*. This therefore excludes giving values to parameters in a block data program unit. Add "a named constant or" after "Only".	241:23
Do we need to add procedure pointer to the list?	243:16-17
There is no constraint in 7.5.2 against an internal procedure being the *target* in a pointer assignment statement. Do we need to prohibit it there, or say here that a procedure pointer that is associated with an internal procedure cannot be used as an actual argument associated with a dummy procedure? It's probably also necessary to say that the *target* in a pointer assignment statement can't be an internal procedure if the *pointer-object* is not declared in the host scoping unit of the internal procedure.	243:28
We say "type parameters (if any)" but not "intent ... if any". Is there any reason not to be consistent? Also, there is no apparent reason for "(if any)" to be a parenthetical remark.	244:14-15
Why is "size" set in small type? Assumed size arrays and assumed-length characters aren't deprecated.	244:19
"... array that is not allocatable or a pointer" can be read "array that is a pointer or is not allocatable," which is wrong. Replace "not allocatable or" by "neither allocatable nor."	244:27
If "entity in the host scoping unit" doesn't include entities accessed by use or host association some extra words are needed.	246:38
Add "The type T could not be moved to another module, and accessed therefrom by use association within the interface body, because its components would then not be accessible."	248:39
Add "and it is accessible" after "one".	248:47

Is it permitted to define an operator to be both unary and binary, by having some one-argument functions and some two-argument functions in its interface block? If not, it would be useful to say so here. If so, an example showing it would be informative. E.g. add something like
`FUNCTION GRAPH_CLOSURE ( ARG ) ! * is unary prefix in this case`
`  IMPORT :: GRAPH`
`  TYPE(GRAPH), INTENT(IN) :: ARG`
`  TYPE(GRAPH) :: GRAPH_CLOSURE`
`END FUNCTION GRAPH_CLOSURE`
at [250:7+].
	249:25-32

Add ", by an interface body," after "statement."	252:13
Does it make sense to add VALUE to the list? (Also see remarks for 5.1.2.14 above).	252:21-26

Should it be permitted for the result type of a dummy function or dummy function procedure 12.3.2.3
pointer to have an assumed type parameter? Allowing an asterisk for the length parameter of the character result type of a dummy function is printed in small type at [68:7-8]. If we go so far as to allow a dummy function or dummy function pointer result type to have a type parameter that is a specification expression (not just an initialization expression), I don't see any extra difficulty in the additional step of allowing the result type to be declared to have an assumed type parameter, so long as it is spelled out that the parameter value is assumed from the associated actual argument's function declaration, not somehow assumed into the function associated as an actual argument from the context of its invocation. If it's OK, we also need a constraint at or near [252:37+]+:

Constraint: If *proc-interface* is present and declares the procedure entities to be functions, and the result type has an assumed type parameter, the functions shall be dummy procedures or dummy procedure pointers.

I don't object if this changes the semantics of dummy functions that have character type result with assumed length, because it was apparently proposed that this be deleted altogether.

I agree that the result of a non-dummy function or function procedure pointer should not be permitted to have an assumed nonkind parameter, except for the "grandfathered" case of character length. This is different, however, from the case of deferred type parameters of a function result, which indicate that the function sets those parameters – this is not a problem because such a result is required to be allocatable or a pointer.

The same remarks apply to assumed shape of the result of a dummy function. That is, if we allow it, the shape is assumed from the associated actual argument, not somehow assumed into the function when it is invoked.

Strengthen [291:5] to the level of a constraint by replacing "internal procedure ... function" by 255:37-39
"internal procedure, a statement function, or a specific intrinsic procedure listed in 13.15 and marked with a bullet (•)".

---

(1) It's interesting that FRAME only gave this two-line constraint one line number. (2) Add 256:6
something to it to require that the label could be referenced by a GO TO statement from the point of the procedure reference.

Needs a reference to the definition of "type-compatible" (but I think a formal definition doesn't 257:8
yet exist).

Presumably, pointers that become undefined, e.g. local pointers of the procedure that don't 258:43
have the SAVE attribute, don't remain associated with the actual argument. Insert "that do not become undefined (14.7.6) and that are" after "pointers."

For the same reason as at [258:43] insert "that does not become undefined (14.7.6) and" after 259:5
the first "pointer."

Because 6.3.3.1 is so far away, insert "; if it is allocatable and the associated dummy argument 259:31
is allocated, it is deallocated as though by execution of a DEALLOCATE statement" after "established".

Couldn't this be a constraint if the procedure has explicit interface? 260:30-33

Couldn't this be a constraint? 260:35-41

Need to add something about the label being one that could be accessed by a GO TO statement 261:4
from the point of the procedure reference.

Should "type parameters" be "assumed or deferred type parameters?" 262:14

I don't understand the "but not both" part. Couldn't one DEALLOCATE (A); ALLOCATE (B); DEALLOCATE (B)?	263:18
Since end statements are execution constructs, delete "or with ... construct."	265:22-23
Isn't a procedure defined by an ENTRY statement in a program unit that begins with a FUNCTION statement also a function subprogram? If so, replace by "A program unit that has a FUNCTION statement as its first statement is a **function subprogram**."	265:25
A *language-binding-spec* isn't one of the things in the RHS of R1224. To what does this constraint refer?	266:11-12
Occurrence of a function name within the function, in the case that RESULT is specified, does not necessarily constitute a recursive reference. (1) If it is passed as an actual argument, there's no guarantee it actually gets invoked. (2) If it is pointer assigned to a procedure pointer accessed by host or use association, it might be invoked after the current instance of the function has terminated. Replace "are recursive function references" by "refer to the function."	267:26
Does occurrence of a function name within the function, in the case that RESULT is not specified, necessarily refer to the result value? What about the case of a recursive reference to a function with a scalar result value? After "unit" add ", other than as the *function-name* in a *function-reference* (R1215),".	267:28
After "pointer" add "array or allocatable array".	267:30
Isn't a procedure defined by an ENTRY statement in a program unit that begins with a SUBROUTINE statement also a subroutine subprogram? If so, replace by "A program unit that has a SUBROUTINE statement as its first statement is a **subroutine subprogram**."	268:42
This section is not specifically germane to definition of a subroutine subprogram. As such it should not be in section 12.5.2, or even in section 12.5. It should be in section 12.4, say as section 12.4.4.	269:25-37
To increase precision insert "as a dummy argument" after "appears".	270:34
Should ENTRY statements provide for a BIND attribute?	271:37-39
Is this normative? It seems to apply to the C standard, not the Fortran standard.	271:40
Do we need to say anything here about pointer or allocatable actual arguments suffering from actions taken on dummy arguments? Or, indeed, from actions on any associated entity, e.g. does the type selector in a SELECT TYPE construct suffer from actions taken on the associate name? Do we need to add "or any associated entity" in lots of places in section 14?	14.6.2.1.2-3
Need a section **14.6.2$\frac{1}{2}$ Events that cause allocatable variables to be deallocated**.	353:19+
Does the appearance of a structure component in a nullify statement constitute a variable definition context? If it doesn't, then *pointer-assignment-stmt* should be added to the list at [359:43] and the list item at [359:44] should be deleted. Otherwise, *nullify-stmt* should be moved from the list item at [359:43] to the list item at [359:44]. Compare to [71:22].	359:43-44

3 Edits

Edits refer to 00-007. 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 the indicated line. Remarks for the editor are noted in the margin, or appear between [and] in the text.

Some of these edits cannot be taken literally, e.g. the one at [46:23]. The intent is to give the editor whatever latitude is necessary to "do the right thing."

[Editor: "FOTRAN" ⇒ "FORTRAN".]	xvi:10
[Editor: Replace "of" by "specified by."]	xvi:13
[Editor: Delete "an interface body or" because it's not needed: In 5.1.2.10, it is stated that a name obtains the EXTERNAL attribute if it is a specific procedure name in an interface body.]	2:40
[Editor: "compatability" ⇒ "compatibility"]	3:12
[Replace "listed in 1.5.2" by "noted in Annex B". There are no deleted features listed in 1.5.2.]	3:33
[Editor: "than" ⇒ "from".]	4:16
[Editor: Move to be after [11:9], to put them into alphabetical order.]	11:6
[Editor: "block" ⇒ "body".]	18:34
[Editor: Replace "or use association" by "use association, or construct association."]	19:23
[Editor: Add "or another Fortran processor" after "itself".]	20:8
[Editor: Delete the sentence that begins "Only objects...". It nearly duplicates the constraint three lines above. Also, it says "object" while the constraint says "entity." Although a function result variable is an object, a function isn't, so the sentence is incorrect.]	33:24-25
[Editor: Replace ".TRUE." by "true".]	36:9
[Editor: Replace ".FALSE." by "false".]	36:11
[Editor: "types" ⇒ "attributes".]	41:5
[Editor: Add ", or parameters of the type in which the component is declared," after "thereof".]	43:12
[Editor: Delete the bracket at the end of the line.]	44:4
[Editor: Frame is probably the culprit that inserted an inscrutable "(724)" twice. I don't know what edits to supply to fix it, but I assume you do.]	46:23, 58:37
[Editor: "parameter" ⇒ "parameters".]	59:44
[Editor: Set "attributes" in bold face and add it to the index.]	63:3
[Editor: Delete one of the periods at the end of the sentence.]	63:6
[Editor: Add "ALLOCATABLE or" before "POINTER".]	64:23
[Editor: Delete "PARAMETER," because the VALUE attribute is permitted only for dummy arguments, and there is another constraint against the PARAMETER attribute for dummy arguments.]	65:9
[Editor: Add "ALLOCATABLE or" before "POINTER".]	66:5
[Editor: "compitable" ⇒ "compatible".]	69:40
[Editor: "compatable" ⇒ "compatible".]	69:44
[Editor: "when" ⇒ "if".]	71:24
[Editor: "shape is" ⇒ "bounds are".]	72:44
[Editor: "shape" ⇒ "bounds".]	72:46

[Editor: Replace "allocation ... parameters" by "argument presence, a property of the type or nondeferred type parameters, or allocation status." "Argument presence" is the only new thing; the order has been changed slightly to be parallel to [74:32-33].]	74:27-28
[Editor: Add "nondeferred" before "type".]	74:33
[Editor: "block" ⇒ "body".]	75:42
[Editor: Delete "then" twice.]	76:42, 77:1
[Editor: Replace "(13.15)" by "listed in 13.15 and not marked by a bullet (•)."]	77:7
[Editor: Delete – Unresolved issue 214 seems no longer to be germane to any nearby text.]	78:21-32
[Editor: "if" ⇒ "of".]	83:5
[Editor: Replace ".FALSE." by "false".]	104:9
[Editor: Replace ".TRUE." by "true".]	104:12
[Editor: There is no section 6.4.3.1. To what does unresolved issue 7 refer? If nothing, then maybe it should be removed?]	104:29
[Editor: "array" ⇒ "variable".]	107:17
[Editor: Remove the "a" before "allocatable."]	107:28
[Editor: Start a new paragraph with "For the character..." since the issue of kind type parameters is unrelated to the remainder of the paragraph.]	114:8
[Editor: After "are" add "those of the specific function referenced,"; after "arguments" add ", as specified in 14.1.2.3".]	115:16-17
[Editor: "and" ⇒ "is".]	118:13
[Editor: Remove "or".]	118:16
[Editor: Although it would take the names out of alphabetical order, if TRANSFER and TRIM were exchanged, the previous line would set better.]	120:10
[Editor: Before "from" insert "accessed by use association".]	120:11
[Editor: Add a hyphen at the end of the line.]	137:1
[Editor: Replace ".TRUE." by "true" twice.]	143:37,40
[Editor: Replace ".TRUE." by "true".]	145:29
[Editor: "POINT.3D" ⇒ "POINT_3D"]	153:40
[Editor: "COLOR.POINT" ⇒ "COLOR_POINT"]	153:43
[Editor: The syntax of SELECT TYPE has been changed so that the example is no longer valid. Replace by "SELECT TYPE (A => P_OR_C)"]	154:7
[Editor: "as" ⇒ "when".]	161:7
[Editor: The most common way to refer to a specifier is by giving its syntax, not its informal name. Replace "record" by "REC="".]	168:11
[Editor: "public parameters" ⇒ "named constants".]	171:14
[Editor: "public parameter" ⇒ "named constant".]	171:19

[Editor: There appears to be an extra space after "(10.7.8)".]	175:41
[Editor: "IN" ⇒ "In".]	176:40
[Editor: Add "(9.2.3.1)" after the second "occurs."]	181:32
[Editor: The part from "A derived-type object" to the end of the note applies to normative text at [184:19-21]. Move it to [184:21+].]	183:36
[Editor: Replace ".TRUE." by "true" twice.]	193:14, 18
Editor: Replace ".FALSE." by "false".]	193:15
[Editor: Insert "subsequent" before "list."]	194:36
[Editor: Should "may" be "shall?"]	194:37
[Editor: Are these sentences written in correct standardese?]	198:1-2
[Editor: Insert a comma after the second "record."]	198:8
[Editor: It is usually correct to say "different from" instead of "different than." "Than" implies ordering.]	204:38
[Editor: Should "when" be "if?"]	205:13
[Editor: Add "1X," after the slash.]	213:21
[Editor: "comma" ⇒ "COMMA".]	213:36
[Editor: Replace "k" by "$-k$".]	216:28
[Editor: Replace "5" by "6".]	219:20
[Editor: Replace "processors" by "processor's" (it's possessive, not plural).]	220:38
[Editor: Replace "6" by "7".]	223:4
[Editor: Add a space in "Amodule".]	237:22
[Editor: Add a space between *op* and the left parentheses, twice.]	239:9-10
[Editor: "block" ⇒ "body".]	245:7
[Editor: remove the semicolon after the second "block".]	247:4
[Editor: Add "listed in 13.15 and not marked with a bullet (•)" after "function"	254:26
[Editor: Delete "(13.15)".]	254:36
[Editor: Add "(12.5.2.3)" after "instance".]	255:43
[Editor: Add a right parentheses after "OUT".]	257:9
[Editor: Remove "then".]	258:22
[Editor: Remove "then".]	260:20
[Editor: Remove "or" at [260:31]. Insert ", or a reference to the NULL() intrinsic" after "pointer".]	260:31-32
[Editor: Replace the second "of the invoking subprogram" by "in the invoking scoping unit."]	261:38
[Editor: Add a period after "permitted."]	263:13
[Editor: "33" ⇒ "34".]	266:23, 40

[Editor: "called" ⇒ "invoked".]	273:20
[Editor: "when" ⇒ "if".]	360:7
[Editor: Replace "the vendor ... support" by "the subset of features that is supported is processor dependent." These kinds of things are usually (always?) described by reference to the processor, not the vendor.]	361:35-36
[Editor: Replace ".TRUE." by "true" and ".FALSE." by "false" throughout.]	362:22-25
[Editor: Replace ".TRUE." by "true" and ".FALSE." by "false" throughout.]	430:13-29