Subject: Comments and questions concerning 99-007r1
References: 97-218r2, 98-181, 98-221, 99-007r1, 99-106r2, 99-107, 99-121r1
From: Van Snyder

There is no pretense offered that remarks in this paper constitute complete edits necessary to correct problems or answer questions noted here. Some of the alleged problems may not be problems at all. Some of them should perhaps turn into unresolved issues.

Page and line numbers refer to 99-007r1.

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
[Editor: Add "direct component" to the index.]	40:43
[Editor: Add "ultimate component" to the index.] Would there be a problem in defining ultimate components to be components of intrinsic type, components that have the POINTER attribute, or components that have the ALLOCATABLE attribute but are not currently allocated? This would loosen things up a little bit at 205:19, q.v. Is "ultimate component" expected to be a static thing-o?	41:8
It seems a long way around to refer 13 pages forward to discover that what's under discussion is on the previous page. Re-word the constraint: Constraint: If EXTENDS or EXTENSIBLE is present, neither SEQUENCE nor BIND(C) shall be present.	42:2-3
Is an ELEMENTAL procedure OK? I can't think of a reason why not, but it does bear pondering.	44:21-25, 39-43
We probably need a constraint that all of the nonkind parameters of the passed-object dummy argument shall be assumed.	44:32+
I don't see that "itself" contributes anything here.	45:17
Would read better if "never allowed to be of a subsequently defined type" were "required to be of a previously defined type."	46:33
An example showing default initialization using a type parameter as part of the initialization expression would be useful. This demonstrates that we should eventually generalize nonkind parameters to allow any type. It opens a can of worms to do so now, but it should be on our list of loose ends to clean up in the next revision.	47:30+
Delete the sentence "The *proc-entity-name* ... parent type" as a consequece of the syntax change made by part 2.2 of 99-121r1.	49:29-31
"a" ⇒ "an"	50:12
It would be clearer to use a construction parallel to 52:14-17, e.g. "If the *type-bound-procedure-part* of a type definition contains a PRIVATE statement, the default accessibility of its procedure bindings is PRIVATE; otherwise, the default accessibility of its procedure bindings is PUBLIC. The accessibility of a procedure binding is the default if it is not explicitly declared by an *access-spec* in its *proc-binding*."	52:23-26
Replace by "The accessibility of a type-bound procedure is not affected by a PRIVATE state-	52:33-34

ment in the *data-component-part*; the accessibility of a data component is not affected by a PRIVATE statement in the *type-bound-procedure-part*."

There's not quite enough room for *intrinsic-structure-constructor*.	61:4
The note suggests that a dummy function can have a result with a character length parameter of "*". This is not currently allowed by normative text at 75:24-32, which are specified at 75:23 to be the only ways allowed. In light of the constraint at 75:9-10, it seems the list is incomplete. Also see remarks at 86:1ff.	76:4-5
The phrase "or DEALLOCATE (6.5.3)" should probably be removed. I don't see how a polymorphic object can acquire a concrete type by execution of a DEALLOCATE statement. If this was intended to say something about the dynamic type becoming undefined as the result of execution of a DEALLOCATE statement, 378:17 (q.v.) is probably a better place to say so. Or, maybe, a subsection 14.X should be devoted to definition and undefinition of dynamic type.	77:5-6
Issue 139, at 105:29-35 questions the use of "shall not be defined" at 105:26. If it's incorrect there, it's probably incorrect here, too. I don't think "become defined" works, either, because it's impossible. Any ideas, anybody?	81:25, 30
What is a "direct or indirect result?" I thought results were something that functions produced. Does this mean that the existence of an instance of a scoping unit is a direct or indirect *consequence* of an action in another instance?	82:40
"function" ⇒ "procedure" twice.	84:12-13
It should be stated here whether VOLATILE and EQUIVALENCE interact.	5.1.2.13
Should it be permitted for the result type of a dummy function or dummy function procedure 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 75:9-10. 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 86:28+: 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.	86:1ff

We need to vote whether to accept the vote on specs for procedure pointers (97-218r2), or the vote on edits for procedure pointers (99-106r2), concerning the meaning of an absent interface specification. See arguments concerning 42:20-21 in 99-133 that it should be left as-is.	87:17-28
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 86:1ff above), 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 "inquiry function?" This seems even more nebulous than the term "reference" that is the subject of issue 140.	108:17
These paragraphs are repetitive and could perhaps be combined. E.g.: An *allocate-object* or a bound or type parameter of an *allocate-object* shall not depend on *stat-variable*, *errmsg-variable*, or on the value, bounds, allocation status, or association status of any other *allocate-object* in the same ALLOCATE statement. Neither *stat-variable* nor *errmsg-variable* shall be allocated within the allocate statement in which it appears, nor shall they depend on the value, bounds, allocation status, or association status of any *allocate-object* in the same ALLOCATE statement. [This is the same as at 125:44-46 – I'm just advocating to move it to be with the other paragraph that says a similar thing.]	125:18-19, 42-46
[Editor: delete, as suggested in issue 76. This should close issue 76.]	125:27-40
Can an allocatable entity be other than a variable? Although *entity* is probably correct (except that there's no normative definition for entity), *variable* would be more precise – if it's correct. The same remark applies at 130:23 and 137:42.	126:33
"instance" ⇒ "instances"	127:26
Delete "still"	129:37
In many other languages, bounds are considered to be what we call "nonkind type parameters." I suggest that we bundle together what we now call type, parameters, and bounds, and call that something like "complete type". Use something like "element type" for the type and type parameters of the elements of an array, if we ever need to discuss them while excluding the shape or bounds.	137:1-8
[Editor: Add a citation for NULL() in the index.]	138:3
It should be specified whether intrinsic or defined assignment, or something else, is used when objects of derived type appear in input lists in READ statements, after *name*= in namelist input, or for purposes of the implied copy if a dummy argument has the VALUE attribute. For the latter, intrinsic assignment semantics may or may not be appropriate (it's a new thing). Is defined assignment ever appropriate in the VALUE case?	7.5.1.5-6 pp156-158
I'm curious why "properties" is used here, but "type parameters and bounds" is used in a parallel sentence at 142:1.	143:27
This is the same kind of wording as at 139:40-1, which is the subject of note 131. If one is wrong, the other probably is, too.	143:31-32
"When" ⇒ "If".	157:22
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

I didn't know that expressions "delivered" anything. Shouldn't this say that the value of the expression shall have the pointer attribute?	159:38
Do we need to say anything special about passed-object dummy arguments?	160:2+
If a file is allowed to be opened both for sequential and stream access, it should be allowed to change between sequential and stream access without closing the file – that is, by executing an OPEN statement for an open file.	187:20-21
Insert "(9.2.4)" after "file storage units," at least because it's a forward reference.	188:21
"is" should be "may be". Otherwise, if sentences that begin "for example" are normative, this sentence states it is always possible to reposition stream files, and always possible to write – both of which contradict statements elsewhere. We also need to take care at 188:32 to say that "any file storage unit may be read" providing the file was not opened with `ACTION='write'`.	188:31
It should be made clear that this section applies only to files not connected for stream access.	9.2.3
It should be made clear that this is Fortran's units for measuring file size, not that Fortran's units for measuring file size shall be the same as the platform's units.	191:16-18
It would be more precise if the phrase "on most processors" were replaced by something like "in the floating-point representation model of 13.7.1 if the radix b is 2 or 16."	198:19-20
What is a "base object?"	203:28
Is the problem with allocatable components that they might have allocatable components? The semantics of allocatable are such that there can't be a cycle. This seems more to be a problem of the definition of "ultimate component" than an insurmountable technical difficulty. See also remarks concerning 41:8.	205:19
Section 9.9.2 says "if no such condition occurs, the processor shall not change the value of *errmsg-variable*." This is incompatible with INTENT(OUT). Replace INTENT(OUT) by INTENT(INOUT).	212:33, 45, 213:9, 21
[Editor: Replace ".." by "."]	226:14
Are any of the problems in unresolved issue 3 (207:10-31) reduced by writing "The input/output operation terminates?" Similarly, are any of those problems reduced by writing "The input transfer terminates" at (227:26, 40)?	227:14
RECURSIVE isn't a characteristic. It would seem that RECURSIVE is required to be consistent between a procedure and its interface definition, between a procedure actual argument and dummy procedure, and between a procedure pointer and procedure pointer or target during pointer assignment. It's been this way since Fortran 90. Am I missing something?	266:4-7
After "specific interface," do we need to say "if the specific interface is accessible"?	269:47
'function" ⇒ "procedure" twice.	273:24, 34
[Editor: "dring" ⇒ "during".]	277:20
We probably need to say the same things *vis-a-vis* INTENT here as at 277:34-42. Note that 99-141 is proposing to change 277:37 to try to repair problems with deferred type parameters.	278:39+
Probably need to include that the argument can't have INTENT(IN).	281:22
Probably need to include that the argument can't have INTENT(IN).	283:11
[Editor: Replace "it" by "they" thrice – refers to "type and type parameters".]	286:12

Do we need to say anything here about pointers that are 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

---

(5) The pointer is pointer-assigned to a target that is allocatable but not currently allocated. 372:35+

[Editor: add to the list (not necessarily here):] 376:42+

($9\frac{1}{3}$) If an error, end-of-file, or end-of-record condition occurs during execution of an input/output statement that contains an ERRMSG= specifier the *errmsg-variable* becomes defined.

($9\frac{2}{3}$) If an error condition occurs during execution of an ALLOCATE or DEALLOCATE statement that contains an ERRMSG= specifier the *errmsg-variable* becomes defined.

[Editor: Add the following in the same paragraph. "If it is polymorphic, its dynamic type becomes undefined."] 378:17

Add ERRMSG= to the list. 379:5

Add *errmsg-variable* to the list. 379:7

The note seems to overstate the obvious. If we really must keep it, (v) should be in the list. 390:38