Subject: Miscellaneous items
From: Van Snyder
References: 00-240

Here are several things that may or may not need attention. I don't even offer edits (well, sometimes I offer crappy ones). 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 00-007r3.

Move [3:30-33] to here. 3:19+

Do we need a constraint that the *type-param-value* for a nonkind type parameter shall be an initialization expression in the declaration of an entity that is not a dummy argument? **Malcolm says no** – this would be incompatible with what is possible for non-component objects, e.g. automatic length for character variables. 30:39+

Constraint: An *access-spec* shall appear only if the derived-type definition is within the specification part of a module. 39:14-17

[See edits for [39:42+] and [41:19+] below. The objectionable part is "This constraint ... *type-bound-procedure-part*". Maybe this constraint isn't needed at all. See [72:19].]

Do we want to allow to specify default values for type parameters? **Malcolm says no** – it wasn't in the spec, and it would be irregular to do this but not do default values for optional arguments. 39:34

Constraint: A PRIVATE statement shall appear only if the derived-type definition is within the specification part of a module. 39:42+

Should ASYNCHRONOUS be in the list? 40:5-8

"Each ... shall not" would read better as "No ... shall". Much as I prefer this, Malcolm says the present wording is the style ISO guidlines require. 40:28

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 remarks for [114:7+] below. 40:28-29, 32-34

The note applies specifically to the syntax term *data-component-def-stmt*, not generally to *component-def-stmt*. Better than changing the note, delete it. It says nothing that's not said better by the syntax rule at [40:3] and the constraint at [40:35-36], which are both on the same page! 40:38

The following duplicates [39:14-17] but that is difficult to find because it is far away: 41:19+

Constraint: A *binding-private-stmt* shall appear only if the derived type definition is within the specification part of a module.

See remarks for [39:14-17] above.

We need a better explanation, or maybe an example, of the problem described by David Moore at meeting 154. I remember being concerned by that problem, but I can't remember the details of it, or an example of it. 60:2-3

Either "or *parent-type-name* in an extension type definition" should be added, or the entire sentence should be deleted (because it's covered by the word "entity" in the previous one). 60:21

This constraint seems inconsistent. Assumed or deferred type parameters are no different from type parameter values that are specification expressions. Either delete the constraint, or require all type parameter values (including the ones that correspond to nonkind parameters) to be initialization expressions. **Malcolm says** the second option would be "a knot in the language." 60:27-28

Is an enumeration really a name? I would think that the name of the enumeration is a name. 61:2

Does the interpretation of this sentence depend on what the meaning of "is" is? (Sorry, I couldn't resist.) Is the sentence true even if BIND(C) is not specified? 61:4

Is a name an enumeration? I would think that a name is the name of an enumeration. In any case, it seems that this sentence and the one at [61:2] constitute a circular definition: An enumeration is a type alias name, and a *type-alias-name* is an enumeration. **Malcolm says** it's not perfect, but it's not broken. 61:19

Replace "each *ac-value* expression" by "all *ac-value* expressions". 63:34

Delete "The appearance ... *entity-decl-list*" because it duplicates the constraint at [66:42-43]. 71:43-72:2

Add "in which they are declared" after "module". Even with this, the use of the word "accessed" is questionable. Suppose `module A` contains 72:20

```
type T;    integer, private :: C;    end type T
```

and some other scoping unit contains

```
use A, only: T   ! Unquestionably an access
type(t) :: V
v%c = 0          ! Is this an "access" of the component C?  After all,
                 ! C is a component of V, which is defined here.
```

The reference to V%C is clearly illegal, but I don't think the sentence at [72:20] prohibits it.

Another question: If a component is of a type that has a private component, is that component private? I think the answer is no. Is there enough here and elsewhere that an interp is not invited? Also see remark for [98:35+] below.

Did the specs really say "disassociated"? Almost certainly "disassociated" should be "undefined". 73:13

Is the description of ASYNCHRONOUS adequate? Suppose a variable V in a common block /B/ 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 /B/ becomes undefined. (1) Should V have the ASYNCHRONOUS attribute? (2) Would the ASYNCHRONOUS attribute prevent /B/ from becoming undefined? /B/ 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. 79:11 ff

---

Copy "A named ... association" from [72:6-8]. Better yet, move [72:3-8] and [84:13-16] to 7.1.7. 84:16

Maybe this is "creeping constraintism" but would it be useful? 98:35+

Constraint: If a *part-name* has the PRIVATE attribute, the *data-ref* shall appear within the same module as the definition of the type of the previous *part-name*.

See remarks for [72:20] above.

or *type-param-name*	114:7+
I suggest: 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.	135:23+
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 [135:23+] 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." If this and the change suggested for [135:23+] above are not accepted, at least move table 7.9 to [136:22+]. Also see 00-318. **Malcolm doesn't like** this one or the previous one.	136:20-22
Maybe we should add "or procedure references" – or maybe we should delete "or a defined assignment statement (7.5.1.6)".	140:18
Do we need this again? It is just an anemic version of [77:23-25].	140:21-22
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?	156:34
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* R830a *loop-limits* **is** *scalar-int-expr*, *scalar-int-expr* [, *scalar-int-expr*] **or** *scalar-int-expr*: *scalar-int-expr* [: *scalar-int-expr*]	158:39-40, 41+
[Editor: "components ... comprise" ⇒ "effective items (9.5.2) that result from expanding".]	190:37
I am disturbed that `TYPE (whateveritis)` is normative.	192:31ff
Delete OPTIONAL and INTENT because they're already prohibited in BLOCK DATA subprograms by the constraint at [66:29-30]. Delete PUBLIC and PRIVATE because they're already prohibited in BLOCK DATA subprograms by the constraint at [72:19].	240:17
Do we need to add "accessible" after "entity," or was the intent to restrict IMPORT to work only for entities declared within the scoping unit containing the interface body?	247:11
This paragraph also needs to mention dummy procedures and procedure pointers.	247:12-14
It would be convenient to be able to use any accessible explicit interface to declare the interface for a procedure pointer. Could we add "**or** *procedure-name*" as an additional right-hand side for R1212? We would also need to replace "consists ... pointers" by "specifies an explicit specific interface, the declared procedures or procedure pointers have the same explicit specific	252:35+

interface" at [253:24-25].

It is not generally necessary for actual arguments associated with INTENT(OUT) dummy arguments to be defined when the procedure reference takes place. In order to have a dynamic type, however, the *data-ref* has to be defined. Do we need a note to the effect that *data-ref* shall be defined even if the passed-object dummy argument has INTENT(OUT)? Or how about a constraint that the passed-object dummy argument shall not have INTENT(OUT)?	255:12+
The phrase "an elemental intrinsic actual procedure may be associated with a dummy argument that is not elemental" leads one to believe that dummy arguments can be elemental. The part "that is not elemental" should be removed. Three possibilities for what to do next are (1) nothing, (2) add a parenthetic remark "(which cannot be elemental)", or (3) put in a note $12.26\frac{1}{2}$ to the effect that dummy arguments cannot be elemental.	260:5-7
We could get rid of "other than as the argument of the PRESENT intrinsic function" by making the argument of the PRESENT intrinsic function optional.	261:8-19
Subclause 14.1.2.3 has nothing to do with generic procedure references. The title ought to be "Unambiguous generic procedure definitions."	347:11
For consistency, either insert "have" before "the same operator" or delete it from the end of the line.	347:23
The sentence "If a generic..." conflicts with, or at least belongs in [348:30-31].	348:3-5
14.1.2.4.1 takes no account of procedure pointers in generics.	349
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 [250:15], but there is no parallel statement for defined assignment.	349:7-17
Either "procedure" should be "interface" at [349:11], or vice-versa at [349:17].	349:11, 17
recursive (12.5.2.1, 12.5.2.2) A procedure that causes itself or another entry in the subprogram that defines it to be invoked, either directly or indirectly, is recursive.	411:39+