Subject: Miscellaneous items
From: Van Snyder
References: 00-179

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.

Given the stipulation at [342:4] that a type name is a name in class (1), and the requirement at [342:13-15] that a name in one class cannot be used to identify another entity in the same class, the "nor ... *type-name*" part of the constraint is unnecessary. 42:6-7

Does PRIVATE default accessibility apply to the parent subobject and components inherited from the parent? 42:27

Does PRIVATE default accessibility apply to the procedures inherited from the parent? 44:14

Shouldn't the constraint be the following? 63:20-21

Constraint: In a *declaration-type-spec*, every *type-param-value* that corresponds to a nonkind type parameter shall be a *specification-expr*, and every *type-param-value* that corresponds to a kind type parameter shall be an *initialization-expr*.

There is no such thing as a "type statement." At these two places, "type statement" should be "type declaration statement." 66:43,48

Shouldn't this paragraph be a constraint? 82:9-12

Add "of type integer" at the end of the constraint. 83:39

There are numerous problems here. (1) A disassociated pointer cannot be referenced in an expression and therefore cannot be a primary. The discussion and table therefore do not belong here. (2) If the pointer is a procedure pointer, it does not have "type, type parameters and rank", and therefore the heading of the second column of table 7.2 does not apply. (3) The description of the result characteristics in the case of *proc-binding* does not include the case when an abstract interface is explicitly specified in the PROCEDURE statement. 115:28-43

Recursive specification functions, e.g. `FACTORIAL`, are useful and not, in themselves, harmful. It seems the real problems occur if specification functions cause recursive invocations of the procedures in which they are contained. Even this isn't guaranteed to be a problem, however: Invoking the specification function might cause the procedure containing the expression to be invoked with different arguments from the instance in which the invocation of the specification function occurred, which might result in the specification function being invoked with different arguments, which might result in termination. One can wrap a reference to a recursive function with a nonrecursive one, and reference the nonrecursive one in the specification expression. Within a specification function, recursive or not, there is nothing to prevent referencing a procedure in which the function appears in a specification expression. Therefore, prohibiting recursive specification functions does nothing to prevent nonterminating recursive invocations of the procedures in which they appear in specification expressions. Since careful programming can prevent a nonterminating recursive invocation, there is probably no need for any kind or prohibition. Specification functions are just another sharp knife in a large armory full of sharp knives, swords and spears. Recursive specification functions are potentially useful, and not intrinsically dangerous. The prohibition at [118:3] against recursive specification functions 118:3

should be removed.	
I suggest: Constraint: In the case of intrinsic assignment, the types of *variable* and *expr* shall conform according to the rules in table 7.9. Put table 7.9 here, but with "type parameters" for a derived type replaced by "kind type parameters." Constraint: In the case of intrinsic assignment, the *variable* and *expr* shall have the same rank or the *expr* shall be a scalar.	132:20+
In concert with the change suggested for [132:20+] above, replace by "If *variable* is an array, *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 [132:20+] above are not accepted, at least move table 7.9 to [133:21+].	133:19-21
The note is repetitive, and in that the part from "A derived-type object" to the end of the note applies to normative text at [183:36], it is confusing.	183:4-12
We find here "A user-defined derived-type input/output procedure is any procedure...." Do we intend to allow internal and dummy procedures, or procedure pointers? The sentence has other problems as well: It isn't enough for a procedure to have the appropriate interface; it needs to be specified in the appropriate interface block. The sentence doesn't contribute anything that's not said elsewhere in the section. Delete it. If not, at least make it consistent with the constraint at [246:30-31]. Also note that one of the proposals in paper 00-179 is to replace the interface-block-based derived-type input/output procedure specification by one based on type-bound procedures.	189:26
Everything in 11.1.2 is said elsewhere, frequently as a constraint. Can we delete section 11.1.2?	237:42-45
Erik Kruyt's paper 00-191 suggests that there is a need to define "identifier" here.	240:20
Is this a "constant" (well, really "nonconstant") that we're trying to get rid of?	245:27
Not needed – it's covered by 14.1.2.3.	246:35-36
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?	246:39
The "otherwise" part is not true for abstract interface blocks.	247:3-4
We may want to point out in a comment that because argument B1 has assumed shape and argument B2 does not, a non-contiguous array section can be the actual argument associated with B1, but a non-contiguous array section cannot be the actual argument associated with B2.	250:4-5
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 R1211? We would also need to replace "consists ... pointers" by "and specifies an explicit specific interface, the declared procedures or procedure pointers have the same explicit specific interface" at [253:7].	252:19+
"R1201" is the wrong syntax rule number.	259:16
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)	260:9-10

nothing, (2) add a parenthetic remark "(which cannot be elemental)", or (3) put in a note 12.27$\frac{1}{2}$ to the effect that dummy arguments cannot be elemental.

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. 261:12-23

I think the reason for this condition is to provide bounds for the elemental-ness. If so, this condition is too strong (the dummy argument of the elemental procedure can't be optional), and not strong enough (the specified array doesn't necessarily provide the desired bounds). It should be "... unless an array of the same rank that is (1) not a dummy argument or is a present dummy argument, (2) not an unallocated allocatable array, and (3) not a disassociated pointer, is supplied as an actual argument of that elemental procedure." 261:15-17

There is no such thing as a "type statement." The "type statement" should be "type declaration statement." 267:17

"pointer" needs to be "pointer or procedure pointer". 324:1

There is at least one, and maybe two problems here. In the phrase "correspond by name to a dummy argument not present" does "not present" mean "not declared," or "it has the optional attribute and there is no associated actual argument?" I think it's the former, but we do have a section with the phrase "dummy arguments not present" in its title – and it refers to the latter. The wording should be revised to avoid this confusion. In the former case, it is impossible for a nonoptional dummy argument to correspond by name to a dummy argument not present. The dummy argument that is not present clearly doesn't have a name. 343:34-35

The sentence "If a generic..." conflicts with, or at least belongs in [344:25-26]. 343:42-44

Not needed, because of [344:40] and the new language in 5.1.2.10 that specifies that an interface body confers the EXTERNAL attribute. Perhaps [344:40] should be re-worded "(d) if there is an explicit specification of the EXTERNAL attribute (5.1.2.10) in that scoping unit". 344:35

A generic interface can be constructed by several interface blocks, in different scoping units. The phrase "in that interface block" is therefore incorrect. Is it enough to delete "block"? 345:7,12-13

Consider the following: 389:42+

```
TYPE(C_PTR) :: Y
REAL(C_FLOAT), ALLOCATABLE, TARGET :: Z(:)
ALLOCATE ( Z(10) )
Y = C_LOC(Z)
DEALLOCATE ( Z )
! Is Y still defined here?  Do we need something in 14.7.6?
```

---

generic interface needs to be in the glossary. 402:10+