Subject: Miscellaneous items
From: Van Snyder
References: 00-192r1

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 00-007r2.

Should the syntax rule number be R4xx, not R314?	32:34
Shouldn't the constraint currently at [55:1-2] be here?	32:39+
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.	41:18-19
Do we want to allow to specify default values for type parameters?	41:32
Needs a new constraint (note that 00-234 adds it): Constraint: If => *binding* appears, the double-colon separator shall appear.	43:22+
[Editor: The Frame-ism (724) here probably ought to be (7.5.1.5).]	45:38
Shouldn't the constraint be the following? 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*.	63:20-21
It's not obvious why the BIND attribute necessarily implies the SAVE attribute. It wouldn't hurt to have an explanation.	79:32-33
Shouldn't this paragraph be a constraint?	82:17-20
Add "of type integer" at the end of the constraint.	84:17
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. There can be no problem except by a mutual recursion of the specification function and the procedure in which it is contained, which requires them both to be recursive. Even this very rare thing isn't guaranteed to be a problem: Invoking the specification function might cause the procedure containing the reference 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. Although prohibiting recursive specification functions prevents nonterminating recursive invocations of the procedures in which they appear in specification expressions, careful programming can do so just as well. There is probably no need for any kind of prohibition. Specification functions are just another sharp knife in a large armory full of sharp knives, swords and spears. The prohibition at [118:3] against recursive specification functions should be removed. If some	118:3

kind of paternalistic prohibition is really desired, it is enough to require that the specification function and the procedure in which it is referenced as a specification function shall not both be recursive.

I suggest: 132:20+

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.

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." 133:19-21

If this and the change suggested for [132:20+] above are not accepted, at least move table 7.9 to [133:21+].

"same type parameters" needs to be "same dynamic type and type parameters as *expr*", and we need to add "for objects whose declared type is the same as the dynamic type of the component of *expr*" if we want to cover the case of polymorphic components. Or do we want to prohibit intrinsic assignment for objects that have polymorphic components? If so, we need to add "and has no ultimate components that are polymorphic and allocatable" after "polymorphic" at [132:33]. But we just added ALLOCATE (... SOURCE = ...), for which the compiler is expected to do assignment. It would seem to be better to mimic ALLOCATE (*variable%component*, SOURCE = *expr%component*) during intrinsic assignment than to prohibit intrinsic assignment if the *variable* has a polymorphic and allocatable ultimate component. If we prohibit intrinsic assignment for objects that have polymorphic and allocatable ultimate components, we also need to prohibit ALLOCATE (... SOURCE = ...) for objects that have polymorphic and allocatable ultimate components. 134:42, 44

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. 156:34

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 the later point [183:36], it is confusing. 183:4-12

The term **effective item** is set in bold-face type, but is in neither the index nor the glossary. 183:42

Everything in 11.1.2 is said elsewhere, frequently as a constraint. Can we delete section 11.1.2? 238:1-4

To what does "the expression" refer? 244:31

Should polymorphicity be a characteristic? 244:33+

Is this a "constant" (well, really "nonconstant") that we're trying to get rid of? 245:36

Not needed – it's covered by 14.1.2.3. 246:46-47

Do we need to add "accessible" after "entity," or was the intent to restrict IMPORT to work 247:3

only for entities declared within the scoping unit containing the interface body?	
This paragraph also needs to mention dummy procedures and procedure pointers.	247:4-6
The "otherwise" part is not true for abstract interface blocks.	247:15-16
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 without causing additional performance problems, but that if a non-contiguous array section is the actual argument associated with B2, a copy operation may be necessary.	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].	252:12+
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)?	254:26+
"If ... polymorphic" could be constraints, because allocatable and pointer dummy arguments require explicit interface.	257:24-28
"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) 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.	260:3-5
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.	261:6-17
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, even if another one provides the bounds), and not strong enough (the specified array doesn't necessarily provide the desired bounds – e.g. if it's unallocated). 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."	261:9-11
"pointer" needs to be "pointer or procedure pointer".	324:1
Subclause 14.1.2.3 has nothing to do with generic procedure references. The title ought to be "Unambiguous generic procedure definitions."	343:10
There is at least one, and maybe three 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	343:34-35

dummy argument that is not present clearly doesn't have a name. The term "consistent" is used. This should be defined more precisely, in terms of characteristics of actual and dummy arguments.

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

These paragraphs do not explicitly apply to defined operations, defined assignment, or user-defined derived-type input/output. 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 or user-defined derived-type input/output. 345:4-14

A generic interface can be constructed by several interface blocks, in different scoping units. The term "interface block" is therefore incorrect. I don't think removing "block" is an adequate repair. 345:7,12-13

Either "procedure" should be "interface" at [345:8], or vice-versa at [345:14]. 345:8,14

The introduction to the list (at [358:10-12]) specifies that it is discussing what happens to "variables local to its scoping unit or local to the current instance of its scoping unit" when a RETURN or END statement is executed. Item (e) discusses the effect on variables that are *not* "local to its scoping unit or local to the current instance of its scoping unit." Either the introduction to the list should be changed, or paragraph (e) should be put somewhere else. Be careful, because 14.6.2.1.3(3) refers here. 358:19-21

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 to say something in 14.7.6?
```

Needs to refer also to 12.3.2.1.2. 400:25

Needs to refer also to 12.3.2.1.1. 400:27

---

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

It is surely an error that the index location for "type specifier" is "67-??". 468