To: J3

Date: 21st March 2001

Subject: Type parameter names and object names

From: Malcolm Cohen

1. Problems with *object-name*

Our default syntax rule "*xyz-name* **is** *name*" has resulted in some very misleading BNF usage.

In particular, there is no definition of *object-name*, but it is generally understood to mean the name of a data object. This is not true - it can be the name of anything. For example,

R602 *designator* **is** *object-name*

(no constraint) and

R601 *variable* **is** *designator*

Constraint: *designator* shall not be a constant or a subobject of a constant.

seem to imply that (e.g.) a module procedure name is a variable, even outside of the module procedure, since it can be produced by the syntax and is not constrained against. This is silly.

We should provide an explicit rule which defines *object-name* and constrain it to be the name of a data object.

2. Other *designator* problems

The definition of *structure-component* omits to specify that the leftmost *part-name* be the name of a data object, though since the standard goes off the rails a few lines later (when describing the semantics) if it is not, one might reasonably guess that this should be the case.

3. Problems with type parameter names

Type parameters are not data objects, nor do I think we wish them to be. But we do wish them to be allowed in expressions inside of the derived-type definition. Currently, they are allowed only because of the excessive looseness of the BNF "*object-name*"; once that is fixed, we need to have specific BNF to permit them in an expression.

Taking this approach, we would not allow "%kind" to be used to discover the kind of a type parameter (because "%kind" can only be applied to objects). The user can use the KIND intrinsic (if available) or simply repeat the expression by which he specified the kind of the type parameter.

3. Problems with procedure pointers

Procedure pointers can have their INTENT and OPTIONAL attributes specified either in the PROCEDURE statement or in the INTENT and OPTIONAL statements. However, due to wording deficiencies, they cannot have their SAVE and POINTER attributes specified in SAVE and POINTER statements. We should fix this inconsistency.

This would not be necessary if we considered procedure pointers to be data objects. Oh well.

The descriptions of how procedure pointers are called and passed as arguments also seem not to be bulletproof.

3. Edits to 01-107

[64:10] Change "entity-decl" to "*entity-decl*".

{Italicise BNF term reference.}

[64:11+] Insert "R505a *object-name* **is** *name*

Constraint: *object-name* shall be the name of a data object."

{Make sure that object designators always refer to objects.}

[79:16+] Insert J3 note:

"Unresolved issue:

We ought to allow procedure pointers to be saved by a SAVE statement as well as by the SAVE attribute on the PROCEDURE statement. Similarly we ought to allow their pointerness to be specified on the POINTER statement."

[81:24,26] Change "*object*" to "*entity*" twice.

{Fix non-sequitur: [81:30] expects this BNF to provide entities, not objects.}

[94:42+] Insert

"Constraint: The leftmost *part-name* shall be the name of a data object."

{Only allow component selection on data objects, not statement functions etc.}

[108:7+] Insert "**or** *type-param-name*

Constraint: *type-param-name* shall be the name of a type parameter."

{Allow type parameters to appear as primaries in an expression. The scoping rules ensure that it is an accessible type parameter, i.e. that we are inside a relevant type definition.}

[253:9+] Insert J3 note:

"Unresolved issue:

The usage of v*ariable* in R1216, R1217 and R1220 w.r.t. procedure pointers is downright misleading if not totally incorrect. This ought to be fixed and simplified."