1 {This shows how the new subclause in the proposal would be typeset. It is followed by other minor edits. Five additional lines would be added, four words and a cross reference would be added to other lines, one line deleted, three constraints deleted, and two constraints added, in other subclauses, for a net increase of 37 lines of normative text and nine lines of notes. The new constraint C441a might be considered to be redundant to C550, and if so need not be added. The last two exceptions in C441b are not needed if the revision of variable definition context proposed in 14-139r1 is adopted. With those changes, the net increase would be 34 lines of normative text and nine lines of notes.}

---

2 [67:1- before subclause 4.5.3] Insert a subclause]

4.5.2.5 PROTECTED attribute for types

1 The PROTECTED attribute for types imposes restrictions on the contexts in which objects of those types may appear. A type with the PROTECTED attribute is a protected type.

2 Except within a module where a protected type is defined, or a descendant of that module, a nonpointer variable of that type, and the target of a pointer of that type, are not definable.

C441a (R427) The PROTECTED attribute shall be specified only in the specification part of a module.

C441b (R427) Except within the module where a protected type is defined, or a descendant of that module, a variable of that type, or that has a subobject of that type, shall not appear in a variable definition context (16.6.7), except as

- the *variable* in a defined assignment, provided the subroutine that defines the assignment is defined in the module where the type is defined, or a descendant of that module,
- the *variable* in a statement specifier, provided the *variable* is required to be of a protected type that is defined in an intrinsic module,
- an actual argument corresponding to a dummy argument that has INTENT(INOUT),
- an *allocate-object* in an ALLOCATE statement without a SOURCE= specifier, or in a DEALLOCATE statement,
- a *data-pointer-object* in a *pointer-assignment-stmt*, or
- a *pointer-object* in a *nullify-stmt*.

C441c (R427) Except within the module where a protected type is defined, or a descendant of that module, a nonpointer subobject of a variable of that type shall not appear

- in a variable definition context (16.6.7),
- as an actual argument corresponding to a dummy argument that does not explicitly have INTENT(IN),
- as the *data-target* in a pointer assignment statement,
- as the *expr* corresponding to a component with the POINTER attribute in a *structure-constructor*,
- as an actual argument corresponding to a dummy argument with the POINTER attribute, or
- as an actual argument in a reference to the C_LOC function from the ISO_C_BINDING intrinsic module.

C441d (R427) Except within the module where a protected type is defined, or a descendant of that module, a pointer subobject of a variable of that type shall not appear in a pointer association context (16.6.8), or as an actual argument corresponding to a pointer dummy argument that does not explicitly have INTENT(IN).

C441e (R425) If EXTENDS appears and the type being defined has a potential subobject component of protected type, its parent type shall either be a protected type, or shall have a potential subobject component of a protected type, and that type shall be defined in the same module as the type of the protected potential subobject component.

3 The value of the actual argument associated with the CPTR argument of the C_F_POINTER subroutine from

the ISO_C_BINDING intrinsic module shall not be the C address of an object of protected type, unless the object is type compatible with the actual argument corresponding to the FPTR argument.

NOTE 4.22b

One can use a pointer to examine objects of protected type, for example to traverse a list or tree, but not to change their values or the pointer associations of their subobjects.

NOTE 4.22c

Constraint C441e ensures that protection cannot be subverted using polymorphism.

NOTE 4.22d

The target of a pointer subobject is not a subobject (6.4.2). Therefore, although it is not possible to change the pointer association status of a pointer subobject of an object of protected type, it is possible to associate a pointer with the same target, or to change the value of its target.

[6:19+] Define a new term

1.3.35.2a
potential subobject component
a nonpointer component, or a potential subobject component of a nonpointer component

---

[63:28+ R427] Insert an alternative to R427:

or PROTECTED

[64:5 C435] After "ABSTRACT" insert "or PROTECTED".

[64:9-11 C438] Delete constraint C438.

[103:2 C555] Replace "procedure pointer or variable" with "derived type definition, procedure pointer, or variable".

[129:11+ C644+] Insert a constraint:

C644a (R630) If any *allocate-object* is unlimited polymorphic, *type-spec* shall not specify a protected type or a type that has a potential subobject component of protected type, and the declared type of *source-expr* shall not be a protected type or a type that has a potential subobject component of protected type.

[160:10+ C716+] Insert a constraint:

C716a (R733) If *data-pointer-object* is unlimited polymorphic, the declared type of *data-target* shall not be a protected type, nor shall it have a potential subobject component of protected type.

[279:29+ 12.5.2.2p1(2)(c)+] Insert a list item

(c′) is of a protected type, or has a subobject of a protected type,

---

[402:15 13.8.2.16p1] Replace "derived type" with "protected derived type (4.5.2.5)".

[402:16-17 13.8.2.16p1] Delete "Therefore it does not have the BIND attribute, and is not a sequence type." {It was proposed to add this to TS 18508, but the proposal was rejected. Should we be consistent?}

[403:24-29 C1303, C1304] Delete constraints C1303 and C1304.

[436:4,6 C1501, C1503] Combine the constraints and add ABSTRACT and PROTECTED:

C1501 (R425) A derived type with the BIND attribute shall not have the ABSTRACT, EXTENDS, SEQUENCE or PROTECTED attribute.