Subject: Provide for the existence of intrinsic derived types
From: Van Snyder

1 Rationale

It may in the future be desirable to define intrinsic derived types. One proposal, to define COMPLEX to be an intrinsic derived type (see 05-262), depends upon this. Others, such as the proposal for a "file handle" alternative to the unit number, might benefit from it. It has been proposed to allow to create new types from existing ones. If a new type is created from an existing intrinsic type, it is necessarily a nonintrinsic type but a not a derived type. Even if none of these things are done, drawing the distinction between nonintrinsic and derived types is harmless, other than the work required to do it.

2 Detailed specification

The intent of this paper is to change terminology without changing any syntax or semantics.

- Classify types along two axes: Intrinsic vs. program defined, and elementary (or some equivalent term) vs. derived. The only purpose of this discussion is to make it clear that "derived" doesn't imply "nonintrinsic."
- Allow intrinsic derived types to exist without requiring definitions to appear within programs.
- Don't include components of intrinsic derived types in the definition of ultimate components.
- Be careful that introducing intrinsic derived types does not introduce contradictions concerning how the following work for existing intrinsic types:
  - equivalence and common,
  - intrinsic, defined, and pointer assignment,
  - I/O list items,
  - Namelist input (namelist output is defined in terms of input, so it needs no special attention, and list-directed needs no attention once list items are done correctly),
  - explicit interface requirements, and
  - C interoperability (this isn't really a problem, but a few edits help to avoid confusion).
- Don't allow user-defined I/O for intrinsic types. Some want to allow this, but it would be a change of semantics, and therefore shouldn't be done as a side effect of this work.
- Allow to override or extend constructors for intrinsic derived types (by not doing anything to prevent it).

3 Syntax

No new syntax, and no changes to existing syntax.

4 Edits

Edits refer to 04-007. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by associated text, while a page and line number followed by + (-) indicates that associated text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

In many of the edits that specify to insert "nonintrinsic" before "derived" it may be better simply to replace "derived" by "nonintrinsic".

[Classify types:] 15:36-37

Types are classified into categories in two ways: Intrinsic as opposed to nonintrinsic types, and elementary as opposed to derived types.

2.4.1.1 Intrinsic and nonintrinsic types

[More of classifying types (new paragraph):] 15:40+

A **nonintrinsic type** is a type defined by a type definition (4.5.1). A nonintrinsic type is necessarily a derived type. Assignment is defined intrinsically (7.4.1.3) for all types, but there are no intrinsic operations for nonintrinsic types. If operations are needed for nonintrinsic types, they shall be supplied as procedure definitions.

[More of classifying types:] 16:5-12

2.4.1.2 Elementary and derived types

An **elementary type** has no internal structure. Elementary types are necessarily intrinsic.

A **derived type** has an internal structure consisting of components that may be of any type. Derived types may be intrinsic or nonintrinsic types. A scalar object of a derived type is called a **structure** (5.1.1.1). Derived types may be parameterized. For each derived type, a structure constructor is available to produce values (4.5.9).

[Allow intrinsic derived types to exist without programmers' definitions: Insert "nonintrinsic" before "derived" twice.] 33:6-7

[Don't delete intrinsic operations on intrinsic derived types: Insert "nonintrinsic" before "derived".] 34:6

[Don't require a type definition for intrinsic derived types: "A" ⇒ "For nonintrinsic derived types, a".] 44:19

[Don't include components of intrinsic derived types in the definition of ultimate components: "derived" ⇒ "nonintrinsic".] 44:26

[Don't delete intrinsic operations and assignment for intrinsic derived types: Replace three instances of "derived-type" in 4.5.10, not including the first one, by "entities of nonintrinsic type".] 65:16-17

[Don't require prior definition of intrinsic derived types: In the edit for [75:8] in 05-201r2, insert "non-intrinsic" before "derived".] 75:8

[Don't allow intrinsic derived type names in accessibility statements: Insert "nonintrinsic" before "derived".] 86:9

[Don't require a prior type definition for constants of intrinsic derived type: "named constant or a structure constructor" ⇒ "structure constructor of a nonintrinsic type or a named constant".] 88:28

[Don't subsume definition of intrinsic type objects in equivalence into definition of sequence derived types: Insert "nonintrinsic" before the first "sequence".] 96:21

[Don't change how intrinsic objects work in common: Insert "nonintrinsic" before "derived".] 100:5

[Don't undo intrinsic assignment for intrinsic derived types. Replace the first instance of "derived" by "nonintrinsic" and delete the second one.] 139:2+8 – in Table 7.8

[Don't require kind type parameters to match for intrinsic assignment of intrinsic derived type entities: In the edit for [139:3-] in 05-198r1, insert "nonintrinsic" before "derived".] 139:3-

---

[Ditto: Insert "that is not a numeric intrinsic assignment statement and" before "for which".] 139:8

[Don't allow intrinsic derived-type pointers to get targets from unlimited polymorphic objects. Insert "nonintrinsic" before "derived".] 143:16

[Remove unnecessary sweeping generalization that would be incorrect for intrinsic derived types: "Unformatted ... This exception" ⇒ "This".] 193:8+4-6

[Don't change how intrinsic list items are processed: Insert "nonintrinsic" before "derived".] 193:9

[Don't change how intrinsic list items are processed: "intrinsic or derived types. In the latter case" ⇒"any types. If the type of a derived-type value is nonintrinsic".] 197:38-39

[Don't allow user-defined I/O for intrinsic types: "derived-type objects and values" ⇒"objects and values of nonintrinsic derived types".] 198:32

[Don't allow user-defined I/O for intrinsic types (maybe this should be in the first paper that defines an intrinsic derived type):] 199:23+

C937$\frac{1}{2}$ (R920) The *derived-type-spec* shall not specify an intrinsic type.

[Don't allow user-defined I/O for intrinsic types: Insert "nonintrinsic" before "derived".] 235:3

[Simplify words that prohibit user-defined I/O for intrinsic types: "not of a derived type" ⇒ "of intrinsic type".] 235:10

[No edit here. This allows input of components of objects of intrinsic derived type. If we don't want to do this, insert "nonintrinsic" before "derived".] 243:25

[Don't change how namelist input works for whole intrinsic-type objects: Insert "nonintrinsic" before "derived".] 244:7

[Don't require explicit interface if a dummy argument is of intrinsic derived type: Insert "nonintrinsic" before "derived".] 257:34

[No edit here. By doing nothing, it becomes possible to override or extend constructors for intrinsic derived types.] 261:12

[Don't allow to define assignment between intrinsic-type objects: "derived" ⇒ "nonintrinsic".] 263:10

[Don't create the appearance of requiring the BIND attribute for interoperable intrinsic derived types: Insert "nonintrinsic" before "derived". This could be left alone because it doesn't say "if and only if," but why risk confusing the reader?] 398:2

[No edit here. Don't bother inserting "nonintrinsic" before "derived" here, because an intrinsic derived type that is interoperable will necessarily have interoperable components.] 398:8+2

[Don't create the appearance of requiring the BIND attribute for interoperable intrinsic derived types: Insert "it is an interoperable intrinsic type or" before "the derived-type definition". This could be left alone because it doesn't say "if and only if," but why risk confusing the reader?] 398:9