Subject: Issues 19 and 211, more work on derived types and constructors
From: Van Snyder
References: 00-247

1 Introduction

Subclause **4.5.6 Construction of derived-type values** doesn't work for extended types. This paper defines the term "component order" and uses it for value construction and intrinsic input/output. It also eliminates the terms "flattened form" and "nested form" that are the subject of issue 211.

Subclause **4.5.3.1 Inheritance** includes a definition of the order of components and parameters of an extended type, for purposes of derived-type value construction and intrinsic input/output, but doesn't define terms by which to reference these definitions. Furthermore, putting them in the "Inheritance" subclause makes them hard to find, even with a cross reference. This paper moves those discussions into subclauses that have no other purpose.

The paragraph at [55:3-7] explains that a *type-param-value* is "assigned to the ... type parameter." This is true only in the case that the *type-param-value* is an expression. It is not true in the case that the *type-param-value* is an asterisk or a colon – at least not without a lot of wriggling that simply isn't there. It's better to say here only how the *type-param-value*s correspond to the type parameters.

The explanation of how component values are constructed from the values in the constructor uses the phrase "converted according to the rules of intrinsic assignment to a value that agrees in type and type parameters with the corresponding component ... the shape of the expression shall conform with the shape of the component." Since this doesn't say it does intrinsic assignment, or that it works as if by intrinsic assignment, there's some question how it handles pointer and allocatable components of a derived type component value. This paper explicitly says "as if by intrinsic assignment."

The paragraph on constructing a value for an allocatable component is silent concerning type parameters, especially deferred parameters, and includes the phrase "any other expression that evaluates to an array" at [57:3] – a holdover from the days when the only allocatable entities were arrays.

This paper revises the discussion of construction of values for allocatable components to include deferred type parameters, and so as not to depend on "arrayness" to get "allocatability."

This paper depends on the terms *parent component* and *ancestor component* introduced in paper 00-247.

2 Edits

Edits refer to 00-007r2. 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 immediately following text, while a page and line number followed by + indicates that immediately following text is to be inserted after the indicated line. Remarks for the editor are noted in the margin, or appear between [and] in the text.

[Editor: Delete. The substance is re-inserted by the edit for [54:28+] below. This is an unex- 52:44-49 pected place to find this material, terms are needed for these concepts, and subclause numbers that subsume only this material are needed for cross references.]

4.5.3$\frac{1}{2}$ Derived type component order 54:28+

[Editor: Insert "order, component" into the index.]

The **component order** of the components of a derived type is the component order of the parent type, if the type is an extended type, followed by the order of the declarations of components declared by *component-def-stmts* in the derived type definition.

The component order of the ultimate components of a derived type is the order of the ultimate components of the parent type, if the type is an extended type and the parent type has components, followed by the order of the declarations of components that are of intrinsic type, and the order of the ultimate components that result from *component-def-stmts* that specify derived type, taken in the order the declarations appear in the derived type definition.

4.5.3$\frac{2}{3}$ Type parameter order

[Editor: Insert "order, type parameter" into the index.]

The **type parameter order** of a derived type is the type parameter order of the parent type, if the type is an extended type and the parent type has parameters, followed by the order of the declarations of parameters declared in the derived type definition.

If the first *type-param-spec* does not include a keyword, it corresponds to the first type parameter in type parameter order (4.5.3$\frac{2}{3}$), and each succeeding *type-param-spec* that does not include a keyword corresponds to the succeeding type parameter in type parameter order. If a keyword is present, the *type-param-value* corresponds to the type parameter named by the keyword. 55:3-7

R450 *component-spec*	**is**	[*keyword* =] *component-constructor*	55:14-18
R450a *component-constructor*	**is**	*expr*	
	or	*target*	

Constraint: Every *type-param-value* within the *derived-type-spec* shall be a *scalar-int-expr*.

Constraint: No component of the type shall have more than one corresponding *component-spec*.

Constraint: There shall be a *component-spec* that corresponds to each component that does not have default initialization.

Constraint: If a *component-constructor* corresponds to a nonpointer nonallocatable component, its type, kind type parameters and rank shall conform to those of the corresponding component in the same way that the *expr* shall conform to the *variable* in an intrinsic assignment (7.5.1.4).

Constraint: If a *component-constructor* corresponds to an allocatable component, the *target* shall be a reference to the intrinsic function NULL() with no arguments, or the type and kind type parameters of the *component-constructor* shall conform to those of the corresponding component in the same way that the *expr* shall conform to the *variable* in an intrinsic assignment (7.5.1.4), the rank of the *component-constructor* shall be the same as the rank of the component, and every deferred type parameter of the *component-constructor* shall correspond to a deferred type parameter of the component. 55:25-32

J3 internal note

Unresolved integration issue xxx
Two of the above constraints refer to intrinsic assignment, wherein the referenced requirements on type, kind type parameters and rank are not constraints.

Constraint: If a *component-constructor* corresponds to a component that has the POINTER attribute, the *target* shall satisfy the same constraints as required of a *target* in a pointer assignment statement (7.5.2) in which the component is the *pointer-object*.

Each *type-param-value* within the *derived-type-spec* (4.5.5) is assigned to the corresponding type parameter as if by intrinsic assignment (7.5.1.5).

If the first *component-spec* does not include a component name keyword and has the same type as an ancestor component, it corresponds directly to the ancestor component having the same type, and therefore indirectly to the components of that type. Each succeeding *component-constructor* that does not include a keyword corresponds to a succeeding component after the last component of that ancestor component, in component order (4.5.3 $\frac{1}{2}$).

Otherwise if the first *component-spec* does not include a component name keyword it corresponds to the first component in component order, and each succeeding *component-constructor* that does not include a keyword corresponds to a succeeding component, in component order (4.5.3$\frac{1}{2}$).

If a component name keyword is present, the *component-constructor* corresponds directly to the component named by the keyword.

If a *component-constructor* corresponds to a nonpointer nonallocatable component, its array bounds if any and its nonkind type parameters shall conform to those of the corresponding component in the same way that the *expr* shall conform to the *variable* in an intrinsic assignment, and its value is assigned to the component as if the *component-constructor* and the component were the *expr* and *variable* in an intrinsic assignment (7.5.1.4).

If a *component-constructor* corresponds to a pointer component, the *target* shall satisfy the same requirements as the *target* in a pointer assignment in which the component is the *pointer-object*, and the *target* is assigned to the component as if the *target* and component were the *target* and *pointer-object* in a pointer assignment (7.5.2).

[Editor: "*expr*" ⇒ "*component-constructor*".] 55:33

[Editor: Delete. This also deletes unresolved issue notes 19 and 211.] 55:35-56:2

[Editor: Delete – superceded by last paragraph of edit for [55:25-32] above.] 56:27-29

If a *component-constructor* corresponds to an allocatable component, the allocation status of the component is as described in Table 4.1. 56:40-57:5

Table 4.1: Allocation status of allocatable component of constructor

component constructor	Allocation status of component
intrinsic function NULL()	not currently allocated
allocatable entity	allocation status of the component constructor
nonallocatable entity	currently allocated

If the *component-constructor* is not a reference to the intrinsic function NULL(), nondeferred type parameters of the component shall have the same values as corresponding type parameters of the *component-constructor*. If the *component-constructor* is currently allocated or not allocatable, the component has the same bounds as the *component-constructor*, the deferred type parameters of the component have the same values as corresponding type parameters of the *component-constructor*, and the value of the *component-constructor* is assigned to the component as if the *component-constructor* and the component were the *expr* and the *variable* in an intrinsic assignment (7.5.1.5).

NOTE 4.48$\frac{1}{2}$

A *component-constructor* that is a pointer and that does not correspond to a pointer component shall be associated with a target.

[Editor: "components ... comprise" ⇒ "effective items (9.5.2) that result from expanding".] 188:44

component order (4.5.3$\frac{1}{2}$): The order of all of the *components* of a *derived type*. 399:23+

type parameter order (4.5.3$\frac{2}{3}$): The order of all of the *type parameters* of a *derived type*. 407:24+