Subject: Define “component order” term, issues 17-19 and 211, more work on constructors
From: Van Snyder
References: 00-148, 00-152

1 Introduction

In paper 00-148, Malcolm addressed issues 17-19 and 211. In paper 00-152, I addressed the definition of the “component order” term. This paper combines those two papers.

Concerning issues 17-19 and 211, Malcolm wrote in paper 00-148:

> Issue 17 says
>
> > “... I’m bothered by having a component name that isn’t the name of a component. Perhaps we should use a different terminology such as subobject name ....”
>
> I concur.
>
> Issue 18 says
>
> > “Should the above not be a constraint? Fix up Grandparents.”
>
> The answer to the question is “No,” but it ought to be part of our scoping rules (which do have similar status as constraints in requiring violation to be diagnosed).
>
> I concur with the second commandment.
>
> Issue 19 says
>
> > “but the name ... is not a component ....”
>
> ok, ok already
>
> Issue 211 says
>
> > “ ‘flattened form’ is used ... but ... nowhere defined”
>
> I concur.

Paper 00-148 introduced the term “subobject name.” This paper instead expands on the definition of “subobject”, which is defined only superficially at [16:23-28]. The term “subobject name” then follows from the term “subobject.”

Section **4.5.6 Construction of derived-type values** doesn’t work for extended types. Section **4.5.3.1 Inheritance** defines the order of components of an extended type, for purposes of derived-type value construction and intrinsic input/output, but doesn’t define the term.

This paper defines the term “subobject order” for nonextensible, base and extended types, and uses the term for value construction and intrinsic input/output.

2 Edits

Edits refer to 00-007r1. 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.

Ultimately, a nonextensible or base derived type is resolved into **ultimate components** that are either of intrinsic type or have the ALLOCATABLE or POINTER attribute. An extended type may be resolved into **ultimate subobjects** (4.5.3.1) if subobjects of the parent subobject are to be included, or ultimate components if subobjects of the parent subobject are not to be included. 41:21-22

[Editor: Delete "For purposes..." to the end of the paragraph.] 53:15-18

[Define a term for the parent subobject.] 53:22-24
An extended type has a **parent subobject** with the type and type parameters of the parent type, consisting of all of the subobjects inherited from the parent type. The name of the parent subobject is the parent type name.

[Editor: Delete issue 17. We no longer call the name of the parent subobject the "component name."] 53:25-34

[Editor: Replace "subobject denoted by the parent type name" with "parent subobject name". (Improve readability by using the newly coined term.)] 53:35

[Editor: Insert a new paragraph. Add this instance of "subobject" to the index.] 53:37+
The **subobjects** of a nonextensible type or of a base type are its components. The subobjects of an extended type are the parent subobject, subobjects of the parent subobject, and the additional components declared, if any.

The **ultimate subobjects** of a nonextensible type or of a base type are its ultimate components. The ultimate subobjects of an extended type are the ultimate subobjects of the parent subobject, and the ultimate subobjects of additional components declared, if any.

> This extends the definition of the term "subobject" and thereby defines the term "subobject name." We use "subobject" instead of "component" when we want to include parent subobject(s). Note that it is defined recursively so that "grandparent" subobjects are included.

Note to J3

[Editor: Replace "have neither" with "not have". Replace "accessible component" with "accessible subobject". Delete "nor ... type". Make the whole thing a note. (Use our new terminology; make it a note because it will be covered by the scoping rules in section 14.)] 53:38-40

[Editor: Delete issue 18.] 53:41-43

4.5.3$\frac{1}{2}$ Derived type subobject order

55:0+

[Editor: Insert "subobject order" into the index.]

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

The subobject order of the ultimate subobjects of a derived type is the order of the ultimate subobjects of the parent subobject, if the type is an extended type and the parent type has subobjects, followed by the order of the declarations of components that are of intrinsic type, and the ultimate subobjects that result from declarations of components of the derived type, taken in the order the declarations appear in the derived type definition.

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

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.

[Editor: "list" ⇒ "order".] 55:22

The structure constructor for any derived type may be in **flattened form**, in which values may be provided for subobjects inherited from the parent type, if any. The structure constructor for an extended type may be in **nested form**, which allows providing a single value for the parent subobject. 55:29+

Constraint: The type name and all subobjects of the type shall be accessible in the scoping unit containing the structure constructor. 55:32-56:4

Constraint: In the flattened form, there shall be at most one *component-spec* corresponding to each subobject of the type other than the parent subobject and no *component-spec* corresponding to the parent subobject. In the nested form, there shall be at most one *component-spec* corresponding to the parent subobject, and at most one *component-spec* corresponding to each component declared for the extended type.

Constraint: In the flattened form, there shall be exactly one *component-spec* corresponding to each subobject of the type, other than the parent subobject, that does not have default initialization. In the nested form, there shall be exactly one *component-spec* corresponding to the parent subobject of the type, and exactly one *component-spec* corresponding to each component declared for the extended type that does not have default initialization.

Constraint: The *keyword* = may be omitted from a *component-spec* only if the *keyword* = has been omitted from each preceeding *component-spec* in the constructor.

Constraint: In the flattened form, each *keyword* shall be the name of a subobject of the type. In the nested form, each *keyword* shall be the name of a component declared for the extended type, or the name of the parent subobject.

If the first *component-spec* has no keyword and the type of the *expr* is the same as the parent type, or if there is a *component-spec* with a keyword that is the same as the parent subobject name, the constructor is in nested form. Otherwise, the constructor is in flattened form.

In the nested form, in the absence of a component name keyword, the first *expr* is assigned to the parent subobject, the second *expr* is assigned to the first component declared in the derived type definition, and each subsequent *expr* is assigned to the sequentially corresponding component declared in the derived type definition.

In the flattened form, in the absence of a component name keyword, each *expr* is assigned to the corresponding subobject of the type, with the subobjects taken in subobject order ($4.5.3\frac{1}{2}$).

If the keyword is the same as the parent subobject name, the *expr* is assigned to the parent subobject; otherwise the *expr* is assigned to the subobject named by the keyword.

[Note to Editor: This includes deleting issues 19 and 211.] 56:7-20

The value that corresponds to the parent subobject is assigned to the parent subobject using intrinsic assignment.

For nonpointer components, the corresponding value is assigned to the corresponding subobject using intrinsic assignment (7.5.1.4).

The previous semantics were "converted according to the rules of intrinsic assignment to a value that has the same type and type parameters as the corresponding component. The shape of the expression shall correspond to the shape of the component." Since this didn't say it did intrinsic assignment, there's some question how it handles pointer and allocatable components of a derived type component value. The revision clarifies this, and also allows a scalar *expr* to be assigned to an array component. *Note to J3*

For pointer components, the corresponding *expr* shall evaluate to an object that would be an allowable target for such a pointer in a pointer assignment statement (7.5.2), and it is assigned to the component using pointer assignment.

[Editor: Delete.] 57:1-3

[Editor: Replace "component" with "subobject". (This now includes subobjects inherited from the parent type in the case of objects of extended type.)] 87:41

[Editor: Replace "component" with "subobject". (This now includes subobjects inherited from the parent type in the case of objects of extended type.)] 89:3

[Editor: Replace "component" with "subobject". (This now includes subobjects inherited from the parent type in the case of objects of extended type.)] 91:10

[Editor: Replace "name of a component" with "subobject name". (Make parent subobjects usable).] 96:37

[Editor: Replace "components" with "subobjects". (This now includes subobjects inherited from the parent type in the case of objects of extended type.)] 103:44

[Editor: Replace "component ultimately in the object" with "ultimate subobject". (This now includes subobjects inherited from the parent type in the case of objects of extended type.)] 183:29-30

[Editor: Replace "component" with "subobject" twice. (This now includes subobjects inherited from the parent type in the case of objects of extended type.)] 183:34

[Editor: Replace "in the same ... unless" by "in the subobject order (4.5.3$\frac{1}{2}$) of the ultimate subobjects unless." 183:38-39

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

[Editor: Replace "components, and binding names" with "bindings, and named subobjects". (Move scoping requirements from section 4 to section 14).] 342:5

---

ultimate subobject (4.5.3): For a *derived type* or a *structure*, a *subobject* that is of *intrinsic type*, has the ALLOCATABLE attribute, or has the POINTER attribute, or an *ultimate subobject* of a subobject that is of *derived type* and does not have the ALLOCATABLE attribute or the POINTER attribute. 407:22+

[Editor: Insert a new paragraph:] 416:24+

A subobject of a nonextensible type or of a base type is the same as a component. A subobject of an extended type is the parent subobject, a subobject of the parent type, or a component of the extended type. The distinction between an ultimate component and an ultimate subobject is that an ultimate subobject might arise from the parent subobject, whereas an ultimate component cannot. Consider the following example:

```
TYPE, EXTENSIBLE :: POINT
  REAL :: X, Y
```

```
END TYPE POINT

TYPE, EXTENDS(POINT) :: PERSON_POINT
  TYPE(PERSON) :: WHO
END TYPE COLOR_POINT
```

The only component of `PERSON_POINT` is `WHO`. The subobjects of `PERSON_POINT` are `X, Y, POINT` and `WHO`. The ultimate subobjects of `PERSON_POINT` are `X, Y, NAME` and `AGE`.