References: 97-209r1 (rationale), 97-256 (specifications), 98-138 (syntax)

The motion passed at meeting 147 was to use the alternative placement and to insert only section 6.4 [now 6.3] (without its subsections) with the addition of a J3 note referencing this paper as early draft of the remaining edits.

The Main Edit

101:24+ Insert new section. [Alternative: 95:44+ Insert new section 6.3, renumbering existing section 6.3 to 6.4.] "

6.4 Initialization and Finalization

Initialization is the process by which the status of a data entity is established prior to any explicit operations on that entity. **Finalization** of a data entity is the processing that follows the explicit operations on that entity. Every data entity in a program undergoes both initialization and finalization, but the possibility exists that initialization or finalization for a particular data entity involves no additional operations.

6.4.1 When initialization and finalization is performed

6.4.1.1 Categorization of data entities for initialization and finalization

For purposes of describing when initialization and finalization is performed, the data entities in a scoping unit may be placed in the following categories:

(1) Data objects made accessible by host association

Initialization and finalization of such data objects is performed according to the categorization of these data entities in the host. Their accessibility in this scoping unit in no way affects when initialization or finalization is performed.

(2) Data objects made accessible by use association

Initialization and finalization of such data objects is performed according to the categorization of these data entities in the module being referenced. Initialization of data objects in a given category in a module occurs before initialization of data objects of that category in all scoping units referencing that module. Finalization of data objects in a given category in a module occurs after finalization of data objects of that category in all scoping units referencing that module.

(3) Dummy variables

Dummy variables are associated with data objects whose finalization will take place later, so no finalization is performed on the dummy variables as such. Dummy variables not of INTENT(OUT) are similarly not subject to the initialization process.

Dummy arguments of INTENT(OUT) receive no status from the associated actual argument and thus are subject to initialization at the same time as category (8).

(4) Function result variable

The function result variable is associated with the function result in the scoping unit invoking the function. The function result is subject to initialization and finalization, so the function result variable is not separately subject to those processes.

(5) Data objects not in the above categories but in a blank common block or a named common block with SAVE attribute

Initialization of data objects in this category occurs once before initialization of data objects in category (7) in all scoping units containing that common block. Finalization of data objects in this category occurs once after finalization of data objects in category (7) in all scoping units containing that common block.

(6) Data objects not in the above categories but in named common block without SAVE attribute

Initialization of data objects in this category occurs before initialization of data objects in category (8) in all scoping units containing that common block. Finalization of data objects in this category occurs after finalization of data objects in category (8) in all scoping units containing that common block.

< processor option to combine?>

(7) Data objects not in the above categories but having the SAVE attribute

Initialization of data objects in this category occurs once before initialization of data objects in category (8) in the scoping unit in which they are declared. Finalization of data objects in this category occurs once after finalization of data objects in category (8) in the scoping unit in which they are declared.

(8) Data objects not in the above categories and not having the SAVE attribute

Initialization and finalization of data objects in this category occurs once for each instance of the scoping unit. For executable program units, this corresponds to executions of the scoping unit. For modules, this is as described in 5.1.2.5.

[Comment: The cited description is added by the minor edits later in this paper.]

(9) Data objects dynamically allocated and deallocated

Initialization and finalization for data objects with the POINTER or ALLOCATABLE attribute occurs normally, but initialization of the storage allocated for such an

object in an ALLOCATE statement is performed as part of the ALLOCATE statement, and finalization of the storage deallocated from such an object in a DEALLOCATE statement is performed as part of the DEALLOCATE statement.

(10) Data entities that are not data objects

Initialization and finalization of anonymous data entities such as function results, structure constructors, etc. is performed as needed.

6.4.1.2 Initialization/finalization ordering within categories

Within a category, initialization of data objects is performed in the order of their first appearance in the declarations of the scoping unit. Finalization is performed in the reverse of this order.

6.4.2 How initialization is performed

Initialization of a data entity consists of two steps, performed in order:

(1) An intrinsic initialization, defined by the language, is applied to the data entity.

(2) A programmed initialization, reflecting the specific program, may then modify that initial state.

Note:

Because initialization consists of steps performed in order, the effect of later steps may, in effect, override the effects of earlier steps. There is no requirement that the successive steps be made manifest if the final effect can be directly determined.

6.4.2.1 Intrinsic initialization

The intrinsic initialization for a data entity with the POINTER attribute causes it to have a pointer association status (14.6.2.1) of undefined.

The intrinsic initialization for a data entity with the ALLOCATABLE attribute causes it to have an allocation status (6.3.1.2) of not currently allocated.

The intrinsic initialization for a data entity with neither the POINTER nor the ALLOCATABLE attribute applies the following rules to the entire data entity if it is scalar or separately to each element of the data entity if it is an array:

(1) If the data entity is of the intrinsic type INTEGER, REAL, COMPLEX, or LOGICAL, (the element of) the data entity is undefined.

(2) If the data entity is of the intrinsic type CHARACTER, each character position in (the element of) the data entity is undefined.

(3) If the data entity is of derived type, the following steps are performed in order:

(a) If the type is an extended type, the entire initialization process is applied recursively to the parent component of (the element of) the data entity.

J3 Note:

Richard Maine has expressed concern about this concept of a parent component. When we resolved how we describe this, apply that resolution to this text, too.

(b) The entire initialization process is applied recursively to the explicitly declared components of (the element of) the data entity, in the order those components appear in the type definition.

Note:

The intrinsic initialization of a derived type includes the programmed initialization of its components.

6.4.2.2 Programmed initialization

Programmed initialization consists of the following two steps performed in order:

(1) If the data entity has neither the POINTER nor the ALLOCATABLE attribute and is of a derived type with one or more initial procedures, the appropriate initial procedure, if any, is applied to the data entity.

<put stuff in here about the nature of the implied call, pseudo-elemental application, and alternative application for the := initializer>

(2) <explicit initialization or default initialization applied by intrinsic assignment>

6.4.3 How finalization is performed

If the data entity has the POINTER attribute, finalization of that data entity performs no operations.

If the data entity has the ALLOCATABLE attribute and is not currently allocated, finalization of that data entity performs no operations.

If the data entity has the ALLOCATABLE attribute and is currently allocated, it is deallocated. The deallocation process performs finalization on the elements being deallocated.

If the data entity has neither the POINTER nor the ALLOCATABLE attribute, the following steps are performed:

(1) <final procedure>

(2) <recursive finalization of explicit components in reverse order>

(3) <recursive finalization of parent component>

J3 Note:
We desperately need to add examples to this material

"

Related Minor Edits

38:34-37 Replace the first three sentences of the paragraph with "Initialization in a component declaration specifies **default initialization** for that component. This default initialization contributes to the initialization process (6.4) for objects of that type." [Many of the details here have been incorporated into 6.4.] [Sometimes we talk about default initialization and sometimes we talk about component initialization. Is there a consistent reason for using one term or the other?]

42:31+ <need addition to allow (INITIAL) and (FINAL) as "names" for type-bound procedures>

44:3 Replace "is initially … (14.7.5)" with "is default initialized (6.4)"

44:4-11 Delete sentences after first two. [This material is covered in another way in 6.4.]

44:12-48 Move into 6.4 somewhere.

46:13-22 Move into 6.4 somewhere.

54:35-36 [Relate to 6.4?]

66:7-9 Replace second sentence in the paragraph with "On invocation of the procedure, such a dummy argument undergoes the initialization process (6.4)." [The new complications are all described in 6.4.]

67:17-18 Replace the final sentence in the paragraph with "Because an INTENT(OUT) variable has no access to any part of the previous status of the actual argument, the initialization process is applied to it."

70:7-15 Replace the first two paragraphs of the section:

"If an object has the **SAVE attribute**, a single copy of the object is shared among all instances (12.5.2.3) of the scoping unit in which it appears, allowing association status,

allocation status, definition status, and value established in one instance to be referenced in another instance. Such an object is called a **saved object**.

If an object in an executable scoping unit does not have the SAVE attribute, a separate copy of that object exists for each instance of the scoping unit. For objects in a module that do not have the SAVE attribute, the basis for sharing is that the instance of a module referenced by an executable scoping unit is the one available to all instances resulting from its procedure invocations. Thus, two instances of executable scoping units referencing a module share the same instance if and only if one is the direct or indirect result of the other or both are the direct or indirect result of a third instance that also references that module."

> [In FORTRAN 77, the descriptive model was that variables existed for the life of a program, but that in the absence of a SAVE statement, they became undefined between executions. In Fortran 90, in order to accomodate features such a recursion and automatic array, we switched to the model that in the absence of SAVE, each execution had its own copy of the variable. However, many vestiges of the F77 descriptive model remain. Because the semantics of intial and final procedures are strongly tied to the F90 model, it is helpful to revise some of the text that still reflects the F77 model.]
>
> [The pointer part of this will be handle elsewhere.]

73:21+<section 5.2 contains no description of the effect of =>NULL( ) initialization>

75:24-28 Replace the second sentence in the paragraph:

"For a common block declared in a SAVE statement, its common block storage sequence (5.6.2.1) in an instance of the scoping unit containing the common block is associated with the common block storage sequence for every other instance of a scoping unit containing the common block, thus making the values in those sequences available to all instances. For a common block not having the SAVE attribute, the basis for sharing is that the common block storage sequence for a common block in an instance of an executable scoping unit is associated with all such common block storage sequences in instances resulting from its procedure invocations. Thus, the common block storage sequences in two instances of executable scoping units containing a common block are associated if and only if one instance is the direct or indirect result of the other or both are the direct or indirect result of a third instance that also contains that common block."

77:19-79:44 <Does any of this need to be moved to 6.4?>

86:10+Insert note: "A *common-block-object* must not be of a type with initial or final procedures, since initial and final procedures are type-bound procedures and sequence types do not have type-bound procedures."

97:31-98-11 Delete. [This should be covered in 6.4.]

98:12-48 Move into 6.4 somewhere.

100:12-31 Delete. [This should be covered in 6.4.]

100:32-34 <move into 6.4?>

100:35-44 Move into 6.4 somewhere.

101:11-24 Delete. [This should be covered in 6.4]

242:37-38 Replace final sentence in paragraph with sentence as for 67:17-18 above.

244:10-12 Replace second sentence with "No initialization (6.4) performed on the dummy data object, even if it has INTENT(OUT)."

328:33-41 Replace both items with "(4) The pointer becomes disassociated by the initialization process (6.4)."

329:1-5 Replace both items with "(2) The target of the pointer is finalized (6.4) by means other than deallocating the pointer." [This text covers both (2) and (3). (4) is covered by the pointer ceasing to exist, so it doesn't matter if it becomes undefined; the new instance of the pointer will be undefined anyway.]

332:13-19 Combine the three items in "(1) Variables that are defined by the initialization process (6.4)."

333:25-26 Replace item with "The initialization process (6.4) may cause part or all of an object to become defined."

333:31-40 Delete. [Subsumed by the above.]

334:11-24 Delete. [These are the variables that cease to exist.]

334:49-50 Replace item with "Successful execution of an ALLOCATE statement for a nonzero-sized object causes the object to become undefined unless the initialization process (6.4) defines it."

335:7-8 Replace "except … specified" with "unless is it defined by the initialization process (6.4)"

335:9-10 [Is this needed separate from the statement about the dummy?]

335:14-15 Same replacement as for 335:7-8.

Ω