Issue 311 - Annex B is Obsolete

To: J3
From: Craig Dedo
Date: January 9, 2001
Subject: Issue 311 - Annex B is Obsolete

Issue
It might seem amusing that the Annex on obsolete features is obsolete, but I doubt that most readers of the standard will appreciate the humor. :-) This annex has mostly been ignored while revising material that it refers to. There might be some of this that is still correct, but I wouldn't trust any of it without carefully checking.

I suggest the possibility that it is sufficient to describe the deleted features rather than giving edit diffs to effectively insert them. Specifically, keep B.1.0, but remove the B.1.x subsections. One could argue for expanding the material in B.1 to discuss possible conversions, just as B.2 does. It seems odd that we discuss conversion of the obsolescent features that are still in the language, but not of the deleted features. All we do for the deleted ones is give edits for undeleting them.

While I acknowledge that many vendors will continue to implement the deleted features, they shouldn't need such explicit detail in the standard. Furthermore, we should not spend our time worrying about how to standardize any interactions between deleted features and new ones. If we craft exact edits to insert the deleted features, that's in essence what we will have to do.

Analysis
While acknowledging the merit of fixing the current state of Annex B, there are at least three (3) ways of resolving the issue of detailed edits in Annex B.

1. Keep the current detailed edits and fix them up to be consistent with the current draft of the standard.
2. Delete the detailed edits and only keep the very brief overview at the beginning of section B.1.
3. Replace the detailed edits with a technical specification of each of the deleted features.

I recommend that J3 take a straw vote on which option it prefers. There are substantial advantages and disadvantages of each option. Following is a brief evaluation of each option.

Option 1 has the advantage of detailed specification of the properties and behavior of each deleted feature. Disadvantages include all of the problems that the editor mentioned when he raised this issue.

Option 2 has the advantage of simplicity. If you delete all of the edits, all of the complications that come with those edits go away. Disadvantages include all of the side effects of using the meat-axe approach.

Option 3 has the advantage of informing the public of the technical behavior and properties of the deleted features. It also avoids the complications that come with providing detailed edits. One disadvantage is that it requires some time and effort to craft the language of the supporting detail for each of these features.

I prefer option 3. This option would allow us to explicitly specify the properties and behaviors of the deleted features without getting entangled in the complications that come with detailed edits.

The second issue of the lack of recommendations for conversion of the deleted features also has merit. This paper includes edits to insert such recommendations into part B.1.1.

Edits

Edits are with respect to the 01-007.

Following are edits to add recommendations for conversion of the deleted features to standard-conforming features.

[411:28] Add at the end of the paragraph, "Programmers can achieve the same result by using loop index variables and numeric expressions in the *loop-control* that are integers and multiplying or dividing the respective variables and expressions by real scale factors."

[411:31] Add at the end of the paragraph, "Programmers can achieve the same result by branching to a CONTINUE statement that is immediately after the END IF statement."

[411:34] Add at the end of the paragraph, "Programmers can achieve the same result by writing a message to the appropriate unit followed by reading from the appropriate unit."

[411:39] Add at the end of the paragraph, "Programmers can achieve the same result by using internal procedures instead of the ASSIGNed GOTO statement and by using default character variables to hold valid format specifications instead of the ASSIGNed FORMAT statement."

[412:3] Add at the end of the paragraph, "Programmers can achieve the same result by using character string edit descriptors instead of H edit descriptors."

Following are two sets of edits for resolving the issue of detailed edits in sections B.1.1 - B.1.5. These sets of edits are for options 2 and 3.

<u>Option 2.</u>
[412:6-414:15] Delete the text.

<u>Option 3.</u>
[412:6-414:15] Replace the existing text with the following text.

The following sections give the technical specification and syntax for extending the standard by implementing any of the deleted features.

B.1.1 Real and double precision DO variables

Redefine *loop-control, io-implied-do-control,* and *do-variable* as:

loop-control	**is**	[,] *do-variable = scalar-numeric-expr, scalar-numeric-expr* [, *scalar-numeric-expr*]
io-implied-do-control	**is**	[,] *do-variable = scalar-numeric-expr, scalar-numeric-expr* [, *scalar-numeric-expr*]
do-variable	**is**	*scalar-variable*

Constraint: The *do-variable* shall be a named *scalar-variable* of type integer, default real or double precision real.

Constraint: Each *scalar-numeric-expr* in *loop-control* shall be of type integer, default real, or double precision real.

The initial parameter m_1, the terminal parameter m_2, and the incrementation parameter m_3 are of the same type and kind type parameter as the do-variable. Their values are established by evaluating *scalar-numeric-expr*$_1$, *scalar-numeric-expr*$_2$, and *scalar-numeric-expr*$_3$, respectively, including, if necessary, conversion to the type and kind type parameter of *do-variable* according to the rules for numeric conversion. If *scalar-numeric-expr*$_3$ does not appear, m_3 has the value 1. The value of m_3 shall not be zero.

The iteration count shall be computed using the expression INT((m_2 - m_1 + m_3) / m_3).

B.1.2 Branching to an END IF statement from outside its IF block

It is permissible to branch to an END IF statement from within the IF construct and also from outside the construct.

B.1.3 PAUSE statement

The definition of the PAUSE statement is

pause-stmt is PAUSE [*stop-code*]

Constraint: A pure subprogram shall not contain a *pause-stmt*.

Execution of a PAUSE statement causes a suspension of the execution of the program. Execution shall be resumable. At the time of suspension of execution, the stop code, if any, is available in a processor-dependent manner. Leading zero digits in the stop code are not significant. Resumption of execution is not under of the control of the program. If execution is resumed, the execution sequence continues as though a CONTINUE statement were executed.

B.1.4 ASSIGN, assigned GO TO, and assigned FORMAT

The definitions of the ASSIGN and assigned GO TO statements are:

assign-stmt **is** ASSIGN *label* TO *scalar-int-variable*

Constraint: The label shall be the statement label of a branch target statement or *format-stmt* that appears in the same scoping unit as the *assign-stmt*.

Constraint: *scalar-int-variable* shall be named and of type default integer.

assigned-goto-stmt **is** GO TO *scalar-int-variable* [[,] (*label-list*)]

Constraint: Each label in *label-list* shall be the statement label of a branch target statement that appears in the same scoping unit as the *assigned-goto-stmt*.

Constraint: *scalar-int-variable* shall be named and of type default integer.

Execution of an ASSIGN statement causes a statement label to be assigned to an integer variable. While defined with a statement label value, the integer variable may be referenced only in the context of an assigned GO TO statement or as a format specifier in an input/output statement. An integer variable defined with a statement label value may be redefined with a statement label value or an integer value.

When an input/output statement containing the integer variable as a format specifier (9.5.1.1) is executed, the integer variable shall be defined with the label of a FORMAT statement.

At the time of execution of an assigned GO TO statement, the integer variable shall be defined with the value of a statement label of a branch target statement that appears in the same scoping unit. Note that the variable may be defined with a statement label value only by an ASSIGN statement in the same scoping unit as the assigned GO TO statement.

The execution of the assigned GO TO statement causes transfer of control so that the branch target statement identified by the statement label currently assigned to the integer variable is executed next.

If the parenthesized list is present, the statement label assigned to the integer variable shall be one of the statement labels in the list. A label may appear more than once in the label list of an assigned GO TO statement.

An assigned GO TO statement shall not appear as the last statement in a non-block DO construct.

Execution of an ASSIGN statement causes the variable in the statement to become defined with a statement label value. The appearance of a variable in an ASSIGN statement is a context that implies definition or undefinition of a variable.

When a numeric storage unit becomes defined, all associated numeric storage units of the same type become defined, except that variables associated with the variable in an ASSIGN statement become undefined when the ASSIGN statement is executed.

Execution of an ASSIGN statement causes the variable in the statement to become undefined as an integer.

A reference to a procedure causes part of a dummy argument to become undefined if the corresponding part of the actual argument is defined with a value that is a statement label value.

A variable in an ASSIGN statement shall not appear in a pure subprogram as a variable which is in common or accessed by host or use association, is a dummy argument of a pure function, is a dummy argument with INTENT (IN) of a pure subroutine, or is an object that is storage associated with any such variable.

A format may also be a *scalar-int-variable* that has been assigned the statement label of a FORMAT statement that appears in the same scoping unit. The *scalar-int-variable* shall be of default kind.

B.1.5 H edit descriptor

The definition of the H edit descriptor is:

h-edit-desc	**is**	cH *rep-char* [*rep-char*] ...
c	**is**	*int-literal-constant*
Constraint:		*c* shall be positive.
Constraint:		*c* shall not have a kind type parameter specified for it.
Constraint:		The *rep-char* in the cH form shall be of default character type.

In the H edit descriptor, *c* specifies the number of characters following the H.

The edit descriptors are without regard to case except for the characters following the H in the H edit descriptor and the characters in character constants.

References

01-007, Fortran 2000 Draft
01-102, Changes to List of Unresolved Issues

[End of J3 / 01-116r1]