J3/01-163r1

To: J3

Date: 20th March 2001

Subject: The term "local variable", and other issues

From: Malcolm Cohen

1. Inroduction

The term "local variable" is not defined anywhere in the standard, but is understood to mean that it is a "variable" that is "local" i.e. that is a "local entity".

The term "local entity" is defined in such a way that variables accessed via use and host association, and dummy arguments and variables in COMMON, all fall into this category.

This is contrary to the usage elsewhere in the computing industry (and in other language standards) and leads to much confusion.

This confusion has spread to the F2002 draft; in "14.6.1.3 Host Association", note 14.12, it states [350:30-31]:
"A name that appears in an ASYNCHRONOUS or VOLATILE statement is not necessarily the name of a local variable."
This statement is completely false; a variable name that appears in any statement, including ASYNCHRONOUS or VOLATILE, in a scoping unit is *necessarily* a local entity of that scoping unit (except for statement entities viz implied-do indexes, and construct entities viz FORALL indexes and associate-variables). Obviously the writer of this statement had the usual programming-language term "local variable" in mind rather than the specific Fortran standard usage.

Also, much of the time, the term "local variable" is preceded or followed by verbiage like:
"not a dummy argument or a subobject thereof, and that is not accessed by use or host association"
to cut it down to what people expect when they see "local variable".

2. Proposal

That we define the term "local variable" to mean what everyone expects it to mean, i.e. to exclude
- variables accessed by use or host association
- dummy arguments
- variables in COMMON (or EQUIVALENCEd into COMMON)

And use this new definition to simplify wording in the standard.

There are only 7 uses of local variable in the standard so far. With the new definition, we can extend the usage to other places to simplify wording.

3. Comments on existing usages in 01-007

The first 3 occurrences of "local variable are in 6.3.1.2 and 6.3.3.1; edits are supplied to use the new definition to simplify wording.
The fourth occurrence of "local variable" is in 12.6 where it states [271:41-42]
"Constraint: A local variable declared in the *specification-part* or *internal-subprogram-part* of a pure subprogram shall not have the SAVE attribute."

This constraint is probably incorrect under the old usage - it would rule out declaring a SAVEd common block. It would also rule out declaring a use-associated variable as being VOLATILE or in NAMELIST, though it is hard to see how useful these are.

Anyway, it certainly appears to be correct under the new definition.

The fifth occurrence is in "14.1.2.1 Common blocks" at [342:47+]; it seems either redundant or incomplete anyway.

The whole sentence reads
"A common block name in a scoping unit also may be the name of any local entity other than a named constant, intrinsic procedure, or a local variable that is also an external function in a function subprogram."

Since a common block name is a global entity, and an external function name is a global entity, this is already ruled out by 14.1.1 which says
"A name that identifies a ... common block or external procedure shall not be used to identify any other global entity ...".

If there were felt to be a need to have some exception clause for function result variables that have the same name as an external function, the exception would need to be much wider to cover the case of a local variable name that matched an external procedure name somewhere else in the program!

We can sidestep that issue by rewording the "permission"-style sentence as a requirement, so that's what I'll do. Note: We don't need to give permission to do things anyway, all we need to do is to avoid prohibiting them!

The sixth occurrence is the one in note 14.12 which already assumes the new definition, so it is ok.

The seventh and last occurrence is at the end of C.9.2, where it talking about restrictions being laid down by a Fortran processor onto a non-Fortran processor. It also seems to assume the new definition of local variable.

4. Other issues

The term "nonsaved" is used twice in the standard and nowhere defined (though perhaps it's implicit from "non"+"saved"?).

We should define this term and use it more widely instead of the clumsy "that does not have the SAVE attribute".

Anyway, "unsaved" is better English, so that's what I'll propose.

5. Edits to 01-007

[16:42+] Insert
"A **local variable** of the scoping unit of a module, main program, or subprogram, is a variable that is a local entity of that scoping unit, is not a dummy argument, is not in COMMON, and is not accessed by use or host association."
{This is in "2.4.3.1.1 Variable". We could equally well put it into Ch14.}

[74:35+] Add to end of paragraph:
"An object that does not have the SAVE attribute is called an **unsaved** object."

[102:28] Replace "An allocatable object with the SAVE attribute" by
"A saved allocatable object".
{Use old term to simplify wording.}

[102:34] Replace "allocatable object that does not have the SAVE attribute," by
"unsaved allocatable object".
{Use new term to simplify wording.}

[102:34-36] Replace ", that is a local ... host association," by
"and is a local variable of a procedure or a subobject thereof"
{Simplify wording of first sentence of this paragraph.}

[102:38] Replace
"allocatable object that does not have the SAVE attribute and that is accessed by use association"
by "unsaved allocatable object that is a local variable of a module".
{Say what we mean.}

[105:9] Append "it has the SAVE attribute or is a function result variable or a subobject thereof; otherwise, it is deallocated (as if by a DEALLOCATE statement)."
{This suffices to specify which local variables retain their allocation status.}

[105:10-15] Delete.
{The list is replaced by the new clause at line 9.}

[105:16-20] Replace with
"If an allocated allocatable object is a local variable of a module, and execution of a RETURN or END statement results in no active scoping unit having access to the module, it is processor-dependent whether the object retains its allocation status or is deallocated."
{Specify status for module variables. Reinstate text from F95 at [81:4-7] which requires that the allocatable be deallocated if it does not retain its allocation status.}

[266:24] Replace "nonsaved" with "unsaved".
{Improve language.}

[342:48-343:1] Replace sentence by
"A name that identifies a common block in a scoping unit shall not be used to identify a constant or intrinsic procedure in that scoping unit."
{Reword problematic and confusing sentence. It is now in a parallel form to the usual requirement at [342:9-12].}

[358:1-4] Replace "direct ... SAVE attribute"by
"default-initialized subcomponents of local variables that do not have the ALLOCATABLE or POINTER attribute, and either are saved"
{Simplify wording. NOTE: This subsumes the edit in 01-162; if that paper does not pass, do a different edit instead.}

[359:32] Replace
"a nonsaved local object that is not a dummy argument, is not accessed by use or host association,"
by "an unsaved local variable".
{Use new terms to simplify wording.}

[360:13-24] Replace with
"(3) When execution of an instance of a subprogram completes,
(a) its unsaved local variables become undefined,
(b) unsaved variables in a named common block that appears in the subprogram become undefined if they have been defined or redefined, unless another active scoping unit is referencing the common block,
(c) unsaved nonfinalizable variables in a module become undefined unless another active scoping unit is referencing the module, and"
[360:31-35] Replace with
" (d) unsaved finalizable variables in a module may be finalized if no other active scoping unit is referencing the module - following which they become undefined."
{Use new terminology to shorten 7-item sublist to 4-item sublist. Put the requirement into a positive statement of what happens. Item (g) in the previous version (replaced by item (d) here) was particularly hard to read.}

[405:41] Add new definition to glossary:

"**local variable** (2.4.3.1.1): A variable local to a particular scoping unit; not imported through use or host association, not a dummy argument, and not a variable in common."