To: J3
From: Matthijs van Waveren
Subject: VOLATILE requirement (specs/syntax/edits/example)
Date: March 6, 1998

1. Specification

The VOLATILE attribute and statement specify that the object associated with it may be accessed or changed by a cause from outside the scope of the standard.

2. Syntax

We propose to introduce an attribute form and a statement form. The following is an illustration of the syntax:

- Attribute form:
  REAL, VOLATILE :: A
- Statement form:
  VOLATILE [::] A

3. Description

An object can have the VOLATILE attribute in a specific scoping unit but need not have it in other scoping units. In this, it has the same scoping and inheritance rules as the ASYNCHRONOUS attribute.

It is intended that a VOLATILE variable may be referenced or defined by non-Fortran means during execution of a Fortran program. The Fortran processor must attempt to use the most recent definition when a value is required. Likewise it should attempt to make the most recent Fortran definition available. It is the programmers responsibility to manage the interactions with the non-Fortran processes. Any variable that affects the sequence of storage units associated with a object with the VOLATILE attribute, also needs to be declared with the VOLATILE attribute.

On the use and constraints of the attribute and statement the following. A procedure and a function shall have an explicit interface, if the procedure has a dummy argument that has the VOLATILE attribute. An array with vector-valued subscripts can not be a dummy argument with a VOLATILE attribute. A local variable declared in the *specification-part* or *internal-subprogram-part* of a pure subprogram shall not have the VOLATILE attribute. If the POINTER and VOLATILE attributes are both specified, then the volatility shall apply to the target of the pointer and not to the pointer association. If the PARAMETER attribute is specified, the VOLATILE attribute shall not be specified.

4. List of Edits

- Add to syntax rule R214 (page 10), as follows:

R214 *specification-stmt* **is** *access-stmt*
or *allocatable-stmt*
or *asynchronous-stmt*
or *common-stmt*
or *data-stmt*
or *dimension-stmt*
or *equivalence-stmt*
or *external-stmt*
or *intent-stmt*
or *intrinsic-stmt*
or *namelist-stmt*
or *optional-stmt*
or *pointer-stmt*
or *save-stmt*
or *target-stmt*
or *volatile-stmt*

- Add to section 2.5.4 (page 18):

When a data object with the VOLATILE attribute is given a value via some mechanism outside the scope of this standard, it is considered to be defined.

- Add to syntax rule R503 (page 49), as follows:

R503 *attr-spec* **is** PARAMETER
or *access-spec*
or ALLOCATABLE
or ASYNCHRONOUS
or DIMENSION (*array-spec*)
or EXTERNAL
or INTENT (*intent-spec*)
or INTRINSIC
or OPTIONAL
or POINTER
or SAVE
or TARGET
or VOLATILE

- Add an extra constraint after R506 (page 50):

Constraint: If the PARAMETER, INTRINSIC, or EXTERNAL attributes are specified, then the VOLATILE attribute shall not be specified.

- Add a new section 5.1.2.13 (page 61):
  VOLATILE attribute

An object can have the VOLATILE attribute in a specific scoping unit but need not have it in other scoping units. All objects associated with an object with the VOLATILE attribute also need this attribute.

NOTE
It is intended that a VOLATILE variable may be referenced or defined by non-Fortran means during execution of a Fortran program. The Fortran processor must attempt to use the most recent definition when a value is required. Likewise it should attempt to make the most recent Fortran definition available. It is the programmers responsiblity to manage the interactions with the non-Fortran processes. Any variable that affects the sequence of storage units associated with a object with the VOLATILE attribute, also needs to be declared with the VOLATILE attribute.
END NOTE

If the POINTER and VOLATILE attributes are both specified, then the volatility shall apply to the target of the pointer and not to the pointer association.

- Add a new section 5.2.11 (page 64) and renumber 5.2.11 to 5.2.12:
  VOLATILE statement

R5xx *volatile-stmt* **is** VOLATILE [::] *object-name-list*

The VOLATILE statement declares the VOLATILE attribute for a list of objects.

- Change in section 11.3.2 the text between Note 11.7 and 11.8 to the following (page 198):

The local name of an entity made accessible by a USE statement may appear in no other specification statement that would cause any attribute (5.1.2) of the entity to be respecified in the scoping unit that contains the USE statement, except that it may appear in a PUBLIC or PRIVATE statement in the scoping unit of a module and it may be given the ASYNCHRONOUS or VOLATILE attribute.

- Change (2) (f) in section 12.3.1.1 (page 203) from:

A dummy argument that has the ASYNCHRONOUS attribute, or

to:

A dummy argument that has the ASYNCHRONOUS or VOLATILE attribute, or

- Somewhere, probably in chapter 14, it is needed to clarify the special scoping and inheritance rules for the ASYNCHRONOUS attribute. Namely that it can be different for the "same" variable in different scoping units. This is also valid for the VOLATILE attribute. Thus we can make the edits for both attributes in parallel. See issue 4 in the paper describing the edits for 98-007.

- Add to section 14.7.5 in page 302:

(23) An object with a VOLATILE attribute, that is changed by a cause outside the scope of the

standard, becomes defined with a processor-dependent value.

5. Example of Usage

The following example illustrates the usage of the VOLATILE attribute in the case of Remote Memory Access (RMA). RMA allows one process to specify all communication parameters, both for the sending side and for the receiving side. Since the other process may not know which data in its own memory might be accessed, we need the VOLATILE attribute in order to specify the "volatility" of the data. The example shows how to implement the generic indirect assignment A = B(map), where A, B, and map have the same distribution, and map is a permutation. We assume a block distribution with equal size blocks. The example originates from the MPI-2 Standard Document, Chapter 6 (July 18, 1997, Message Passing Interface Forum), with the addition of the VOLATILE attribute.

```
   SUBROUTINE MAPVALS(A, B, map, m, comm, p)
!
!
   USE MPI
!
! *** Subroutine arguments
!
   INTEGER m                   ! extent of index, target and
                               ! source arrays
   INTEGER map(m)              ! index array
   INTEGER comm                ! communicator, specifies group
                               ! of processes
   INTEGER p                   !
   REAL A(m)                   ! target array
   REAL, VOLATILE :: B(m)      ! source array
!
! *** Local variables
!
   INTEGER sizeofreal          ! size of real in bytes
   INTEGER win                 ! handle to window in memory
                               ! accessible by other processes
   INTEGER ierr                ! error number
   INTEGER i                   !
   INTEGER j                   ! rank of target
   INTEGER k                   ! displacement from window start
                               ! to the beginning of the target
                               ! buffer
!
! *** Executable code
!
! ***
!
```

```
   CALL MPI_TYPE_EXTENT(MPI_REAL, sizeofreal, ierr)
!
! *** Creation of a memory window of size m*sizeofreal by each
! *** process in comm that is accessible by remote processes.
!
   CALL MPI_WIN_CREATE(B, m*sizeofreal, sizeofreal, &
     MPI_INFO_NULL, comm, win, ierr)
!
! *** Synchronisation of Remote Memory Access calls on win within
! *** the group comm.
!
   CALL MPI_WIN_FENCE(0, win, ierr)
!
! ***
!
   DO i = 1, m
     j = map(i)/p          ! calculation of rank of target
     k = MOD(map(i), p)    ! calculation of displacement
!
! *** Data transfer from the target memory [A(i)] to the caller
! *** memory [win, B(map(i))].
!
     CALL MPI_GET(A(i), 1, MPI_REAL, j, k, 1, MPI_REAL, &
          win, ierr)
   ENDDO
!
! *** Synchronisation of Remote Memory Access calls on
! *** win within the group comm.
!
   CALL MPI_WIN_FENCE(0, win, ierr)
!
! *** Freeing of memory windows in each process.
!
   CALL MPI_WIN_FREE(win, ierr)

   RETURN
   END
```