

```

call user1(a,rq)
call MPI_WAIT(rq,status,ierr)
write (*,*) a

subroutine user1(buf,request)
call MPI_IRECV(buf,...,request,...)
end

```

If `a` is copied, `MPI_IRECV` will alter the copy when it completes the communication and will not alter `a` itself.

Note that copying will almost certainly occur for an argument that is a non-trivial expression (one with at least one operator or function call), a section that does not select a contiguous part of its parent (e.g., `A(1:n:2)`), a pointer whose target is such a section, or an assumed-shape array that is (directly or indirectly) associated with such a section.

If there is a compiler option that inhibits copying of arguments, in either the calling or called procedure, this should be employed.

If a compiler makes copies in the calling procedure of arguments that are explicit-shape or assumed-size arrays, simple array sections of such arrays, or scalars, and if there is no compiler option to inhibit this, then the compiler cannot be used for applications that use `MPI_GET_ADDRESS`, or any non-blocking MPI routine. If a compiler copies scalar arguments in the called procedure and there is no compiler option to inhibit this, then this compiler cannot be used for applications that use memory references across subroutine calls as in the example above.

Special Constants

MPI requires a number of special “constants” that cannot be implemented as normal Fortran constants, e.g., `MPI_BOTTOM`. The complete list can be found in Section 2.5.4 on page 14. In C, these are implemented as constant pointers, usually as `NULL` and are used where the function prototype calls for a pointer to a variable, not the variable itself.

In Fortran the implementation of these special constants may require the use of language constructs that are outside the Fortran standard. Using special values for the constants (e.g., by defining them through `parameter` statements) is not possible because an implementation cannot distinguish these values from legal data. Typically these constants are implemented as predefined static variables (e.g., a variable in an MPI-declared `COMMON` block), relying on the fact that the target compiler passes data by address. Inside the subroutine, this address can be extracted by some mechanism outside the Fortran standard (e.g., by Fortran extensions or by implementing the function in C).

Fortran 90 Derived Types

MPI does not explicitly support passing Fortran 90 derived types to choice dummy arguments. Indeed, for MPI implementations that provide explicit interfaces through the `mpi` module a compiler will reject derived type actual arguments at compile time. Even when no explicit interfaces are given, users should be aware that Fortran 90 provides no guarantee of sequence association for derived types or arrays of derived types. For instance, an array of a derived type consisting of two elements may be implemented as an array of the first elements followed by an array of the second. Use of the `SEQUENCE` attribute may help here, somewhat.