Fortran 2015

John Reid, ISO Fortran Convener,
JKR Associates and
Rutherford Appleton Laboratory

BCS Fortran Specialist Group
London, 29 September 2016

Abstract

The content of the next revision was chosen in 2015 (here) so is known informally as Fortran 2015.

It is under construction and was formally approved as a "New Work Item" at the meeting of SC22 in September. A complete draft exists (N2110).

It includes further interoperability with C and further coarray features (Technical Specifications). Also some small changes.

I will summarize the new features and the process that will be needed to complete the standard.

Further interoperability with C

Fortran 2003 provides for interoperability of procedures with arguments that are scalars, explicit-shape arrays, or assumed-size arrays, and are not optional.

It does not provide for arguments that are **assumed shape**, **allocatable**, **pointer**, or **optional**.

Fortran 2015 fills this gap by defining C descriptors for such arguments. They are provided by the system when calling C from Fortran and have to be constructed by C code for calling Fortran.

Fortran 2015 also allows C functions to accept arguments of **any rank** or **any type**.

C descriptor

A C descriptor for an object is a struct of type CFI_cdesc_t with components:

base_addr C address of the first element of the object. NULL if unallocated or not associated.

elem_len The sizeof() an element of the object.

rank Rank of the object.

type Code for the type of the object.

Attribute Code to indicate whether the object is allocatable, a pointer, assumed-shape, or otherwise.

dim[] Lower bounds, extents, and stride multipliers.

The new calling mechanism

A dummy argument in a Fortran interface that is allocatable, assumed-shape, or a pointer may correspond to a formal parameter in a C prototype that is a pointer to a C descriptor.

When calling the C function from Fortran, a suitable C descriptor is provided by the system.

Assumed type and rank

A dummy argument in an interface may be of assumed type and/or rank. E.g.

```
interface
   subroutine archive(a,b)
      type(*) :: a
      real :: b(..)
   end subroutine archive
end interface
```

It may correspond to a pointer to a C descriptor in a C function prototype.

Constructing C descriptors in C

A C descriptor must not be initialized, updated, or copied other than by calling one of these functions.

- CFI_allocate does a Fortran allocation
- CFI_deallocate does a Fortran deallocation
- CFI_establish establishes a C descriptor
- CFI_section updates a C descriptor to describe an ordinary array section
- CFI_select_part updates a C descriptor to describe an array
- section such as array%part
- CFI_setpointer updates a C descriptor to point to the whole of an object or be disassociated.

No mixing of C and Fortran allocation mechanisms is allowed.

Further coarray features: Teams

Needed for independent computations on subsets of images.

Change team construct defines division into subteams and back.

Code that has been written and tested on whole machine should run on a team, so image indices are within team.

Collective activities, including syncs and allocations, are relative to team.

Further coarray features: Collectives

The collective subroutines are:

`co_broadcast, co_max, co_min,`

`co_sum, co_reduce.`

Invoked by the same statement on all images of the team and involve synchronization within them.

Further coarray features: Image failure

There is provision for image failure because of the huge numbers of images likely to be in use.

Once failed, an image remains so.

Can test by adding stat=variable to image control statement or remote reference.

failed_images intrinsic lists failed images in a team.

Further coarray features: Events

Events are useful if one or more images need to do something before another image can continue.

Special type for events

The doers “post” to a coarray of event type and the other “waits” on that coarray.

Further coarray features: More intrinsic atomic subroutines

```
atomic_add
atomic_fetch_add
atomic_cas
atomic_and
atomic_fetch_and
atomic_or
atomic_fetch_or
atomic_xor
atomic_fetch_xor
```

Minor changes

Allow any constant properties of an object to be referenced within its declaration; e.g.

integer :: B = bit_size(B)

Allow specification of locality within DO CONCURRENT

For entities accessed from a module, allow specification of the default for PUBLIC/PRIVATE

Minor changes re I/O

Allow a file to be connected to more than one unit

Allow w=0 in Ew.d, ESw.d, ENw.d edit descriptors, to conform with G0.d

Allow the .d in G0.d to be ignored if the corresponding value is not real or complex

Allow e=0 in Ew.dEe etc, to specify minimal exponent width

Minor changes re procedures

Make RECURSIVE the default for procedures

Extend host association to be able to control both importation and hiding a host entity

Allow IMPLICIT NONE to be used to require EXTERNAL to be specified explicitly

Permit modification of VALUE arguments in a pure function

Procedures in ISO_C_Binding, other than C_F_POINTER, are pure.

Minor changes re intrinsics

New intrinsic functions: REDUCE, COSHAPE, OUT_OF_RANGE

Enhanced functions: CMPLX, SIGN, EXTENDS_TYPE_OF, SAME_TYPE_AS, RANDOM_NUMBER

Improve consistency of:

- specifying KIND for intrinsic arguments
- specifying DIM in intrinsic argument lists
- providing optional ERRMSG arguments

New Obsolescences and Deletions

Deleted:

- Arithmetic IF
- Shared DO termination and DO termination on a statement other than END DO or CONTINUE

Obsolescent:

- EQUIVALENCE
- COMMON & BLOCK DATA
- Labelled DO loops
- Specific names for intrinsic functions
- FORALL construct

Timetable
(ISO Limit: Sept 2019)

Feb 2017	CD determined
Mar 2017	CD ballot initiated
May 2017	CD ballot comments available
Oct 2017	DIS determined
Nov 2017	DIS ballot initiated
Feb 2018	DIS ballot results available
Feb 2018	FDIS determined
Apr 2018	FDIS ballot initiated
Jun 2018	FDIS ballot results available
July 2018	Standard published