

Chapter 1

Errata for MPI-3.0

This document was processed on September 12, 2013.

The known corrections to MPI-3.0 are listed in this document. All page and line numbers are for the official version of the MPI-3.0 document available from the MPI Forum home page at www.mpi-forum.org. Information on reporting mistakes in the MPI documents is also located on the MPI Forum home page.

- In all `mpi_f08` subroutine and function definitions in Chapters 3–17 and Annex A.3, in Example 5.21 on page 187 line 13, and in all `mpi_f08` `ABSTRACT INTERFACE` definitions (on page 183 line 47, page 268 lines 23 and 33, page 273 line 47, page 274 line 9, page 277 lines 12 and 21, page 344 line 22, page 346 line 12, page 347 line 36, page 475 lines 10 and 43, page 476 line 38, page 537 line 29, page 538 line 2, and page 678 line 11 - page 680 line 35), the `BIND(C)` must be removed.

- Section 8.2, page 339 (`MPI_ALLOC_MEM`) line 47 reads

but with a different linker name:

but should read

but with a different specific procedure name:

- Section 8.2, page 340 (`MPI_ALLOC_MEM`) lines 10-11 read

The linker name base of this overloaded function is `MPI_ALLOC_MEM_CPTR`. The implied linker names are described in Section 17.1.5 on page 605.

but should read

The base procedure name of this overloaded function is `MPI_ALLOC_MEM_CPTR`. The implied specific procedure names are described in Section 17.1.5 on page 605.

- Section 11.2.2, page 408 (`MPI_WIN_ALLOCATE`) line 2 reads

but with a different linker name:

1 but should read

2 but with a different specific procedure name:

- 3
- 4 • Section 11.2.2, page 408 (MPI_WIN_ALLOCATE) lines 14-15 read

5
6 The linker name base of this overloaded function is
7 MPI_WIN_ALLOCATE_CPTR. The implied linker names are described in
8 Section 17.1.5 on page 605.

9 but should read

10
11 The base procedure name of this overloaded function is
12 MPI_WIN_ALLOCATE_CPTR. The implied specific procedure names are de-
13 scribed in Section 17.1.5 on page 605.

- 14
- 15 • Section 11.2.3, page 409 (MPI_WIN_ALLOCATE_SHARED) line 33 reads

16 but with a different linker name:

17 but should read

18 but with a different specific procedure name:

- 19
- 20 • Section 11.2.3, page 409 (MPI_WIN_ALLOCATE_SHARED) lines 44-46 read

21
22 The linker name base of this overloaded function is
23 MPI_WIN_ALLOCATE_SHARED_CPTR. The implied linker names are de-
24 scribed in Section 17.1.5 on page 605.

25 but should read

26
27 The base procedure name of this overloaded function is
28 MPI_WIN_ALLOCATE_SHARED_CPTR. The implied specific procedure names
29 are described in Section 17.1.5 on page 605.

- 30
- 31 • Section 11.2.3, page 411 (MPI_WIN_SHARED_QUERY_CPTR) line 14 reads

32 but with a different linker name:

33 but should read

34 but with a different specific procedure name:

- 35
- 36 • Section 11.2.3, page 411 (MPI_WIN_SHARED_QUERY_CPTR) lines 26-27 read

37
38 The linker name base of this overloaded function is
39 MPI_WIN_SHARED_QUERY_CPTR. The implied linker names are described
40 in Section 17.1.5 on page 605.

41 but should read

42
43 The base procedure name of this overloaded function is
44 MPI_WIN_SHARED_QUERY_CPTR. The implied specific procedure names are de-
45 scribed in Section 17.1.5 on page 605.

- Section 14.2.1, page 555 (Profiling interface) lines 38-40 read

For Fortran, the different support methods cause several linker names. Therefore, several profiling routines (with these linker names) are needed for each Fortran MPI routine, as described in Section 17.1.5 on page 605.

but should read

For Fortran, the different support methods cause several specific procedure names. Therefore, several profiling routines (with these specific procedure names) are needed for each Fortran MPI routine, as described in Section 17.1.5 on page 605.

- Section 14.2.7, page 560 (Profiling interface, Fortran support methods) lines 29-32 read

The different Fortran support methods and possible options for the support of subarrays (depending on whether the compiler can support `TYPE(*)`, `DIMENSION(..)` choice buffers) imply different linker names for the same Fortran MPI routine. The rules and implications for the profiling interface are described in Section 17.1.5 on page 605.

but should read

The different Fortran support methods and possible options for the support of subarrays (depending on whether the compiler can support `TYPE(*)`, `DIMENSION(..)` choice buffers) imply different specific procedure names for the same Fortran MPI routine. The rules and implications for the profiling interface are described in Section 17.1.5 on page 605.

- Section 17.1.1, page 598 (Fortran support, overview) lines 29-32 read

The Fortran interfaces of each MPI routine are shorthands. Section 17.1.5 defines the corresponding full interface specification together with the used linker names and implications for the profiling interface.

but should read

The Fortran interfaces of each MPI routine are shorthands. Section 17.1.5 defines the corresponding full interface specification together with the specific procedure names and implications for the profiling interface.

- Section 17.1.2, page 599 (Fortran support through the `mpi_f08` module) lines 19-20 read

Define all MPI handles with uniquely named handle types (instead of `INTEGER` handles, as in the `mpi` module).

but should read

Define the derived type `MPI_Status`, and define all MPI handles with uniquely named handle types (instead of `INTEGER` handles, as in the `mpi` module).

- Section 17.1.2, page 601 (Fortran support through the `mpi_f08` module) lines 11-15 read

The **INTERFACE** construct in combination with **BIND(C)** allows the implementation of the Fortran `mpi_f08` interface with a single set of portable wrapper routines written in C, which supports all desired features in the `mpi_f08` interface. TS 29113 also has a provision for **OPTIONAL** arguments in **BIND(C)** interfaces.

but should be removed.

- Both the last Advice to implementors in Section 17.1.4 (Fortran support through the `mpif.h` include file), page 604 line 29 - page 605 line 11, and the whole Section 17.1.5 (Interface specification, linker names and the profiling interface), page 605 line 29 - page 609 line 31 are replaced with:

17.1.5 Interface Specifications, Procedure Names, and the Profiling Interface

The Fortran interface specification of each MPI routine specifies the routine name that must be called by the application program, and the names and types of the dummy arguments together with additional attributes. The Fortran standard allows a given Fortran interface to be implemented with several methods, e.g., within or outside of a module, with or without **BIND(C)**, or the buffers with or without TS29113. Such implementation decisions imply different binary interfaces and different specific procedure names. The requirements for several implementation schemes together with the rules for the specific procedure names and its implications for the profiling interface are specified within this section, but not the implementation details.

Rationale. This section was introduced in MPI-3.0 on Sep. 21, 2012. The major goals for implementing the three Fortran support methods have been:

- Portable implementation of the wrappers from the MPI Fortran interfaces to the MPI routines in C.
- Binary backward compatible implementation path when switching `MPI_SUBARRAYS_SUPPORTED` from `.FALSE.` to `.TRUE.`
- The Fortran PMPI interface need not be backward compatible, but a method must be included that a tools layer can use to examine the MPI library about the specific procedure names and interfaces used.
- No performance drawbacks.
- Consistency between all three Fortran support methods.
- Consistent with Fortran 2008 + TS 29113.

The design expected that all dummy arguments in the MPI Fortran interfaces are interoperable with C according to Fortran 2008 + TS 29113. This expectation was not fulfilled. The **LOGICAL** arguments are not interoperable with C, mainly because the internal representations for `.FALSE.` and `.TRUE.` are compiler dependent. The provided interface was mainly based on **BIND(C)** interfaces and therefore inconsistent

with Fortran. To be consistent with Fortran, the `BIND(C)` had to be removed from the callback procedure interfaces and the predefined callbacks, e.g., `MPI_COMM_DUP_FN`. Non-`BIND(C)` procedures are also not interoperable with C, and therefore the `BIND(C)` had to be removed from all routines with `PROCEDURE` arguments, e.g., from `MPI_OP_CREATE`.

Therefore, this section was rewritten in an erratum to MPI-3.0. (*End of rationale.*)

A Fortran call to an MPI routine shall result in a call to a procedure with one of the specific procedure names and calling conventions, as described in Table 1.1 on page 5. Case is not significant in the names.

No.	Specific procedure name	Calling convention
1A	<code>MPI_Isend_f08</code>	Fortran interface and arguments, as in Annex A.3, except that in routines with a choice buffer dummy argument, this dummy argument is implemented with non-standard extensions like <code>!\$PRAGMA IGNORE_TKR</code> , which provides a call-by-reference argument without type, kind, and dimension checking.
1B	<code>MPI_Isend_f08ts</code>	Fortran interface and arguments, as in Annex A.3, but only for routines with one or more choice buffer dummy arguments; these dummy arguments are implemented with <code>TYPE(*)</code> , <code>DIMENSION(...)</code> .
2A	<code>MPI_ISEND</code>	Fortran interface and arguments, as in Annex A.4, except that in routines with a choice buffer dummy argument, this dummy argument is implemented with non-standard extensions like <code>!\$PRAGMA IGNORE_TKR</code> , which provides a call-by-reference argument without type, kind, and dimension checking.
2B	<code>MPI_ISEND_FTS</code>	Fortran interface and arguments, as in Annex A.4, but only for routines with one or more choice buffer dummy arguments; these dummy arguments are implemented with <code>TYPE(*)</code> , <code>DIMENSION(...)</code> .

Table 1.1: Specific Fortran procedure names and related calling conventions. `MPI_ISEND` is used as an example. For routines without choice buffers, only 1A and 2A apply.

Note that for the deprecated routines in Section 15.1 on page 591, which are reported only in Annex A.4, scheme 2A is utilized in the `mpi` module and `mpif.h`, and also in the `mpi_f08` module.

To set `MPI_SUBARRAYS_SUPPORTED` to `.TRUE.` within a Fortran support method, it is required that all non-blocking and split-collective routines with buffer arguments are implemented according to 1B and 2B, i.e., with `MPI_Xxxx_f08ts` in the `mpi_f08` module, and with `MPI_XXXX_FTS` in the `mpi` module and the `mpif.h` include file.

The `mpi` and `mpi_f08` modules and the `mpif.h` include file will each correspond to exactly one implementation scheme from Table 1.1 on page 5. However, the MPI library may contain multiple implementation schemes from Table 1.1.

Advice to implementors. This may be desirable for backwards binary compatibility in the scope of a single MPI implementation, for example. (*End of advice to implementors.*)

Rationale. After a compiler provides the facilities from TS 29113, i.e., `TYPE(*)`, `DIMENSION(..)`, it is possible to change the bindings within a Fortran support method to support subarrays without recompiling the complete application provided that the previous interfaces with their specific procedure names are still included in the library. Of course, only recompiled routines can benefit from the added facilities. There is no binary compatibility conflict because each interface uses its own specific procedure names and all interfaces use the same constants (except the value of `MPI_SUBARRAYS_SUPPORTED` and `MPI_ASYNC_PROTECTS_NONBLOCKING`) and type definitions. After a compiler also ensures that buffer arguments of nonblocking MPI operations can be protected through the `ASYNCHRONOUS` attribute, and the procedure declarations in the `mpi_f08` and `mpi` module and the `mpif.h` include file declare choice buffers with the `ASYNCHRONOUS` attribute, then the value of `MPI_ASYNC_PROTECTS_NONBLOCKING` can be switched to `.TRUE.` in the module definition and include file. (*End of rationale.*)

Advice to users. Partial recompilation of user applications when upgrading MPI implementations is a highly complex and subtle topic. Users are strongly advised to consult their MPI implementation's documentation to see exactly what is – and what is not – supported. (*End of advice to users.*)

Within the `mpi_f08` and `mpi` modules and `mpif.h`, for all MPI procedures, a second procedure with the same calling conventions shall be supplied, except that the name is modified by prefixing with the letter “P”, e.g., `PMPI_Isend`. The specific procedure names for these `PMPI_Xxxx` procedures must be different from the specific procedure names for the `MPI_Xxxx` procedures and are not specified by this standard.

A user-written or middleware profiling routine should provide the same specific Fortran procedure names and calling conventions, and therefore, can interpose itself as the MPI library routine. The profiling routine can internally call the matching PMPI routine with any of its existing bindings, except for routines that have callback routine dummy arguments, choice buffer arguments, or that are attribute caching routines (`MPI_{COMM|WIN|TYPE}_{SET|GET}_ATTR`). In this case, the profiling software should invoke the corresponding PMPI routine using the same Fortran support method as used in the calling application program, because the C, `mpi_f08` and `mpi` callback prototypes are different or the meaning of the choice buffer or `attribute_val` arguments are different.

Advice to users. Although for each support method and MPI routine (e.g., `MPI_ISEND` in `mpi_f08`), multiple routines may need to be provided to intercept the specific procedures in the MPI library (e.g., `MPI_Isend_f08` and `MPI_Isend_f08ts`), each profiling routine itself uses only one support method (e.g., `mpi_f08`) and calls the real MPI routine through the one PMPI routine defined in this support method (i.e., `PMPI_Isend` in this example). (*End of advice to users.*)

Advice to implementors. If all of the following conditions are fulfilled:

- the handles in the `mpi_f08` module occupy one Fortran numerical storage unit (same as an `INTEGER` handle),
- the internal argument passing mechanism used to pass an actual `ierror` argument to a non-optional `ierror` dummy argument is binary compatible to passing an actual `ierror` argument to an `ierror` dummy argument that is declared as `OPTIONAL`,
- the internal argument passing mechanism for `ASYNCHRONOUS` and non-`ASYNCHRONOUS` arguments is the same,
- the internal routine call mechanism is the same for the Fortran and the C compilers for which the MPI library is compiled,
- the compiler does not provide TS 29113,

then the implementor may use the same internal routine implementations for all Fortran support methods but with several different specific procedure names. For TS 29113 quality, new routines are needed only for routines with choice buffer arguments. (*End of advice to implementors.*)

Advice to implementors. In the Fortran support method `mpif.h`, compile-time argument checking can be also implemented for all routines. For `mpif.h`, the argument names are not specified through the MPI standard, i.e., only positional argument lists are defined, and not key-word based lists. Due to the rule that `mpif.h` must be valid for fixed and free source form, the subroutine declaration is restricted to one line with 72 characters. To keep the argument lists short, each argument name can be shortened to a minimum of one character. With this, the two longest subroutine declaration statements are

```
SUBROUTINE PMPI_Dist_graph_create_adjacent(a,b,c,d,e,f,g,h,i,j,k)
SUBROUTINE PMPI_Rget_accumulate(a,b,c,d,e,f,g,h,i,j,k,l,m,n)
```

with 71 and 66 characters. With buffers implemented with TS 29113, the specific procedure names have an additional postfix. The longest of such interface definitions is

```
INTERFACE PMPI_Rget_accumulate
SUBROUTINE PMPI_Rget_accumulate_fts(a,b,c,d,e,f,g,h,i,j,k,l,m,n)
```

with 70 characters. In principle, continuation lines would be possible in `mpif.h` (spaces on columns 73-131, & on column 132, and on column 6 of the continuation line) but this would not be valid if the source line length is extended with a compiler flag to 132 characters. Column 133 is also not available for the continuation character because lines longer than 132 characters are invalid with some compilers by default.

The longest specific procedure names are `PMPI_Dist_graph_create_adjacent_f08` and `PMPI_File_write_ordered_begin_f08ts` both with 35 characters in the `mpi_f08` module. For example, the interface specifications together with the specific procedure names can be implemented with

```
MODULE mpi_f08
  TYPE, BIND(C) :: MPI_Comm
  INTEGER :: MPI_VAL
```

```

1      END TYPE MPI_Comm
2      ...
3      INTERFACE MPI_Comm_rank ! (as defined in Chapter 6)
4          SUBROUTINE MPI_Comm_rank_f08(comm, rank, ierror)
5              IMPORT :: MPI_Comm
6              TYPE(MPI_Comm),      INTENT(IN)  :: comm
7              INTEGER,              INTENT(OUT) :: rank
8              INTEGER, OPTIONAL,    INTENT(OUT) :: ierror
9          END SUBROUTINE
10     END INTERFACE
11 END MODULE mpi_f08
12
13 MODULE mpi
14     INTERFACE MPI_Comm_rank ! (as defined in Chapter 6)
15         SUBROUTINE MPI_Comm_rank(comm, rank, ierror)
16             INTEGER, INTENT(IN)  :: comm ! The INTENT may be added although
17             INTEGER, INTENT(OUT) :: rank ! it is not defined in the
18             INTEGER, INTENT(OUT) :: ierror ! official routine definition.
19         END SUBROUTINE
20     END INTERFACE
21 END MODULE mpi

```

And if interfaces are provided in `mpif.h`, they might look like this (outside of any module and in fixed source format):

```

22
23 !23456789012345678901234567890123456789012345678901234567890123456789012
24     INTERFACE MPI_Comm_rank ! (as defined in Chapter 6)
25         SUBROUTINE MPI_Comm_rank(comm, rank, ierror)
26             INTEGER, INTENT(IN)  :: comm ! The argument names may be
27             INTEGER, INTENT(OUT) :: rank ! shortened so that the
28             INTEGER, INTENT(OUT) :: ierror ! subroutine line fits to the
29         END SUBROUTINE ! maximum of 72 characters.
30     END INTERFACE

```

(End of advice to implementors.)

Advice to users. The following is an example of how a user-written or middleware profiling routine can be implemented:

```

31
32
33
34
35 SUBROUTINE MPI_Isend_f08ts(buf,count,datatype,dest,tag,comm,request,ierror)
36     USE :: mpi_f08, my_noname => MPI_Isend_f08ts
37     TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
38     INTEGER, INTENT(IN) :: count, dest, tag
39     TYPE(MPI_Datatype), INTENT(IN) :: datatype
40     TYPE(MPI_Comm), INTENT(IN) :: comm
41     TYPE(MPI_Request), INTENT(OUT) :: request
42     INTEGER, OPTIONAL, INTENT(OUT) :: ierror
43     ! ... some code for the begin of profiling
44     call PMPI_Isend (buf, count, datatype, dest, tag, comm, request, ierror)
45     ! ... some code for the end of profiling
46 END SUBROUTINE MPI_Isend_f08ts

```

Note that this routine is used to intercept the existing specific procedure name `MPI_Isend_f08ts` in the MPI library. This routine must not be part of a module.

This routine itself calls `PMPI_Isend`. The `USE` of the `mpi_f08` module is needed for definitions of handle types and the interface for `PMPI_Isend`. However, this module also contains an interface definition for the specific procedure name `MPI_Isend_f08ts` which conflicts with the definition of this profiling routine (i.e., the name is doubly defined). Therefore, the `USE` here specifically excludes the interface from the module by renaming the unused routine name in the `mpi_f08` module into “`my_noname`” in the scope of this routine. (*End of advice to users.*)

Advice to users. The `PMPI` interface allows intercepting `MPI` routines. For example, an additional `MPI_ISEND` profiling wrapper can be provided that is called by the application and internally calls `PMPI_ISEND`. There are two typical use cases: a profiling layer that is developed independently from the application and the `MPI` library, and profiling routines that are part of the application and have access to the application data. With `MPI-3.0`, new Fortran interfaces and implementation schemes were introduced that have several implications on how Fortran `MPI` routines are internally implemented and optimized. For profiling layers, these schemes imply that several internal interfaces with different specific procedure names may need to be intercepted, as shown in the example code above. Therefore, for wrapper routines that are part of a Fortran application, it may be more convenient to make the name shift within the application, i.e., to substitute the call to the `MPI` routine (e.g., `MPI_ISEND`) by a call to a user-written profiling wrapper with a new name (e.g., `X_MPI_ISEND`) and to call the Fortran `MPI_ISEND` from this wrapper, instead of using the `PMPI` interface. (*End of advice to users.*)

-
- Section 17.1.6, page 610 (MPI for different Fortran standard versions) line 27 reads
The routines are not `BIND(C)`.
but should be removed.
 - Section 17.1.6, page 610 (MPI for different Fortran standard versions) line 33 reads
The linker names are specified in Section 17.1.5 on page 605.
but should read
The specific procedure names are specified in Section 17.1.5 on page 605.
 - Section 17.1.6, page 611 (MPI for different Fortran standard versions) line 21 reads
`BIND(C, NAME='...')` interfaces.
but should be removed.
 - After Section 17.1.6, page 611 (MPI for different Fortran standard versions) line 26, which reads
arguments.
the following list item should be added:

The ability to overload the operators `.EQ.` and `.NE.` to allow the comparison of derived types (used in MPI-3.0 for MPI handles).

- Section 17.1.6, page 611 (MPI for different Fortran standard versions) line 43 reads

The routines are not `BIND(C)`.

but should be removed.

- Section 17.1.6, page 611 (MPI for different Fortran standard versions) line 47 reads

The linker names are specified in Section 17.1.5 on page 605.

but should read

The specific procedure names are specified in Section 17.1.5 on page 605.

- Section 17.1.6, page 612 (MPI for different Fortran standard versions) lines 22-24 read

- `OPTIONAL` dummy arguments are allowed in combination with `BIND(C)` interfaces.
- `CHARACTER(LEN=*)` dummy arguments are allowed in combination with `BIND(C)` interfaces.

but should be removed.

- Section 17.1.7, page 614 (Requirements on Fortran compilers) lines 25-47 read

All of these rules are valid independently of whether the MPI routine interfaces in the `mpi_f08` and `mpi` modules are internally defined with an `INTERFACE` or `CONTAINS` construct, and with or without `BIND(C)`, and also if `mpif.h` uses explicit interfaces.

Advice to implementors. Some of these rules are already part of the Fortran 2003 standard if the MPI interfaces are defined without `BIND(C)`. Additional compiler support may be necessary if `BIND(C)` is used. Some of these additional requirements are defined in the Fortran TS 29113 [41]. Some of these requirements for MPI-3.0 are beyond the scope of TS 29113. (*End of advice to implementors.*)

Further requirements apply if the MPI library internally uses `BIND(C)` routine interfaces (i.e., for a full implementation of `mpi_f08`):

- Non-buffer arguments are `INTEGER`, `INTEGER(KIND=...)`, `CHARACTER(LEN=*)`, `LOGICAL`, and `BIND(C)` derived types (handles and status in `mpi_f08`), variables and arrays; function results are `DOUBLE PRECISION`. All these types must be valid as dummy arguments in the `BIND(C)` MPI routine interfaces. When compiling an MPI application, the compiler should not issue warnings indicating that these types may not be interoperable with an existing type in C. Some of these types are already valid in `BIND(C)` interfaces since Fortran 2003, some may be valid based on TS 29113 (e.g., `CHARACTER*(*)`).
- `OPTIONAL` dummy arguments are also valid within `BIND(C)` interfaces. This requirement is fulfilled if TS 29113 is fully supported by the compiler.

but should read

All of these rules are valid for the `mpi_f08` and `mpi` modules and independently of whether `mpif.h` uses explicit interfaces.

Advice to implementors. Some of these rules are already part of the Fortran 2003 standard, some of these requirements require the Fortran TS 29113 [41], and some of these requirements for MPI-3.0 are beyond the scope of TS 29113. (*End of advice to implementors.*)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48