Annex B

Change-Log

Annex B.1 summarizes changes from the previous version of the MPI standard to the version presented by this document. Only significant changes (i.e., clarifications and new features) that might either require implementation effort in the MPI libraries or change the understanding of MPI from a user's perspective are presented. Editorial modifications, formatting, typo corrections and minor clarifications are not shown. If not otherwise noted, the section and page references refer to the locations of the change or new functionality in this current version of the standard. Changes in Annexes B.2–B.4 were already introduced in the corresponding sections in previous versions of this standard.

B.1 Changes from Version 3.0 to Version 3.1

B.1.1 Fixes to Errata in Previous Versions of MPI

#388 1. Chapters 3–17, Annex A.3 on page 714, and Example 5.21 on page 187, and MPI-3.0 Chapters 3-17, Annex A.3 on page 707, and Example 5.21 on page 187.
   Within the `mpi_f08` Fortran support method, `BIND(C)` was removed from all `SUBROUTINE`, `FUNCTION`, and `ABSTRACT INTERFACE` definitions.

#415 2. Section 3.2.5 on page 30, and MPI-3.0 Section 3.2.5 on page 30.
   The three public fields MPI_SOURCE, MPI_TAG, and MPI_ERROR of the Fortran derived type `TYPE(MPI_Status)` must be of type `INTEGER`.

#424 3. Section 3.8.2 on page 67, and MPI-3.0 Section 3.8.2 on page 67.
   The flag arguments of the Fortran interfaces of MPI_IMPROBE were originally incorrectly defined as `INTEGER` (instead as `LOGICAL`).

#345 4. Section 6.4.2 on page 237, and MPI-3.0 Section 6.4.2 on page 237.
   In the `mpi_f08` binding of MPI_COMM_IDUP, the output argument newcomm is declared as `ASYNCHRONOUS`.

#345 5. Section 6.4.4 on page 248, and MPI-3.0 Section 6.4.4 on page 248.
   In the `mpi_f08` binding of MPI_COMM_SET_INFO, the intent of comm is `IN`, and the optional output argument ierror was missing.

#419 6. Section 7.6 on page 314, and MPI-3.0 Sections 7.6, on pages 314.
   In the case of virtual general graph topolgies (created with MPI_CART_CREATE), the

use of neighborhood collective communication is restricted to adjacency matrices with the number of edges between any two processes is defined to be the same for both processes (i.e., with a symmetric adjacency matrix).

#345
7. Section 8.1.1 on page 333, and MPI-3.0 Section 8.1.1 on page 335.
   In the mpi_f08 binding of MPI_GET_LIBRARY_VERSION, a typo in the resultlen argument was corrected.

#388
8. Sections 8.2 (MPI_ALLOC_MEM and MPI_ALLOC_MEM_CPTR),
   11.2.2 (MPI_WIN_ALLOCATE and MPI_WIN_ALLOCATE_CPTR),
   11.2.3 (MPI_WIN_ALLOCATE_SHARED and MPI_WIN_ALLOCATE_SHARED_CPTR),
   11.2.3 (MPI_WIN_SHARED_QUERY and MPI_WIN_SHARED_QUERY_CPTR),
   14.2.1 and 14.2.7 (Profiling interface), and corresponding sections in MPI-3.0.
   The linker name concept was substituted by defining specific procedure names.

#362
9. Section 11.2.2 on page 405, and MPI-3.0 Section 11.2.2 on page 407.
   The same_size info key can be used with all window flavors.

#350
10. Section 11.3.4 on page 423, and MPI-3.0 Section 11.3.4 on page 424.
    Origin buffer arguments to MPI_GET_ACCUMULATE are ignored when the MPI_NO_OP operation is used.

#355
11. Section 11.3.4 on page 423, and MPI-3.0 Section 11.3.4 on page 424.
    Clarify the roles of origin, result, and target communication parameters in MPI_GET_ACCUMULATE.

#383
12. Section 14.3 on page 567, and MPI-3.0 Section 14.3 on page 561
    New paragraph and advice to users clarifying intent of variable names in the tools information interface.

#383
13. Section 14.3.3 on page 569, and MPI-3.0 Section 14.3.3 on page 563.
    New paragraph clarifying variable name equivalence in the tools information interface.

#383
14. Sections 14.3.6, 14.3.7, and 14.3.8 on pages 573, 580, and 592, and
    MPI-3.0 Sections 14.3.6, 14.3.7, and 14.3.8 on pages 567, 573, and 584.
    In functions MPI_T_CVAR_GET_INFO, MPI_T_PVAR_GET_INFO, and MPI_T_CATEGORY_GET_INFO, clarification of parameters that must be identical for equivalent control variable / performance variable / category names across connected processes.

#391
15. Section 14.3.7 on page 580, and MPI-3.0 Section 14.3.7 on page 573.
    Clarify return code of MPI_T_PVAR_{START,STOP,RESET} routines.

#386
16. Section 14.3.7 on page 580, and MPI-3.0 Section 14.3.7 on page 579, line 7.
    Clarify the return code when bad handle is passed to an MPI_T_PVAR_* routine.

#388
17. Section 17.1.3 on page 609, and MPI-3.0 Section 17.1.4 on page 603.
    The advice to implementors at the end of the section was rwritten and moved into the following section.

#388
18. Section 17.1.5 on page 612, and MPI-3.0 Section 17.1.5 on page 605.
    The section was fully rewritten. The linker name concept was substituted by defining specific procedure names.

#388
19. Section 17.1.6 on page 617, and MPI-3.0 Section 17.1.6 on page 611.
    The requirements on `BIND(C)` procedure interfaces are removed.

#389
20. Annexes A.2, A.3, and A.4 on pages 692, 714, and 763, and MPI-3.0 Annexes A.2, A.3, and A.4 on pages 685, 707, and 756.
    The predefined callback MPI_CONVERSION_FN_NULL was added to all three annexes.

#345
21. Annex A.3.4 on page 731, and MPI-3.0 Annex A.3.4 on page 724.
    In the `mpi_f08` binding of MPI_{COMM|TYPE|WIN}_{DUP|NULL_COPY|NULL_DELETE}_FN, all `INTENT(...)` information must be removed.

B.1.2 Changes in MPI-3.1

#349+ #402+ #404+ #421
1. Sections 2.6.4 and 4.1.5 on pages 20 and 101.
   The use of the intrinsic operators "+" and "-" for absolute addresses is substituted by MPI_AINT_ADD and MPI_AINT_DIFF. In C, they can be implemented as macros.

#357
2. Sections 8.1.1, 8.7, and 12.4 on pages 333, 355, and 484.
   The routines MPI_INITIALIZED, MPI_FINALIZED, MPI_QUERY_THREAD, MPI_IS_THREAD_MAIN, MPI_GET_VERSION, and MPI_GET_LIBRARY_VERSION are callable from threads without restriction (in the sense of MPI_THREAD_MULTIPLE), irrespective of the actual level of thread support provided, in the case where the implementation supports threads.

#369
3. Section 11.2.1 on page 403.
   The same_disp_unit info key was added for use in RMA window creation routines.

#273
4. Sections 13.4.2 and 13.4.3 on pages 509 and 514.
   Added MPI_File_iread_at_all, MPI_File_iwrite_at_all, MPI_File_iread_all, and MPI_File_iwrite_all

#378
5. Sections 14.3.6, 14.3.7, and 14.3.8 on pages 573, 580, and 592.
   Clarified that `NULL` parameters can be provided in MPI_T_{CVAR|PVAR|CATEGORY}_GET_INFO routines.

#377+ #400
6. Sections 14.3.6, 14.3.7, 14.3.8, and 14.3.9 on pages 573, 580, 592, and 596.
   New routines MPI_T_CVAR_GET_INDEX, MPI_T_PVAR_GET_INDEX, MPI_T_CATEGORY_GET_INDEX, were added to support retrieving indices of variables and categories. The error codes MPI_T_ERR_INVALID and MPI_T_ERR_INVALID_NAME were added to indicate invalid uses of the interface.

B.2 Changes from Version 2.2 to Version 3.0

B.2.1 Fixes to Errata in Previous Versions of MPI

1. Sections 2.6.2 and 2.6.3 on pages 19 and 19, and MPI-2.2 Section 2.6.2 on page 17, lines 41-42, Section 2.6.3 on page 18, lines 15-16, and Section 2.6.4 on page 18, lines 40-41.

This is an MPI-2 erratum: The scope for the reserved prefix MPI_ and the C++ namespace MPI is now any name as originally intended in MPI-1.

2. Sections 3.2.2, 5.9.2, 13.6.2 Table 13.2, and Annex A.1.1 on pages 25, 176, 540, and 669, and MPI-2.2 Sections 3.2.2, 5.9.2, 13.5.2 Table 13.2, 16.1.16 Table 16.1, and Annex A.1.1 on pages 27, 164, 433, 472 and 513
This is an MPI-2.2 erratum: New named predefined datatypes MPI_CXX_BOOL, MPI_CXX_FLOAT_COMPLEX, MPI_CXX_DOUBLE_COMPLEX, and MPI_CXX_LONG_DOUBLE_COMPLEX were added in C and Fortran corresponding to the C++ types bool, std::complex<float>, std::complex<double>, and std::complex<long double>. These datatypes also correspond to the deprecated C++ predefined datatypes MPI::BOOL, MPI::COMPLEX, MPI::DOUBLE_COMPLEX, and MPI::LONG_DOUBLE_COMPLEX, which were removed in MPI-3.0. The non-standard C++ types Complex<...> were substituted by the standard types std::complex<...>.

3. Sections 5.9.2 on pages 176 and MPI-2.2 Section 5.9.2, page 165, line 47.
This is an MPI-2.2 erratum: MPI_C_COMPLEX was added to the "Complex" reduction group.

4. Section 7.5.5 on page 302, and MPI-2.2, Section 7.5.5 on page 257, C++ interface on page 264, line 3.
This is an MPI-2.2 erratum: The argument rank was removed and in/outdegree are now defined as int& indegree and int& outdegree in the C++ interface of MPI_DIST_GRAPH_NEIGHBORS_COUNT.

5. Section 13.6.2, Table 13.2 on page 540, and MPI-2.2, Section 13.5.3, Table 13.2 on page 433.
This was an MPI-2.2 erratum: The MPI_C_BOOL "external32" representation is corrected to a 1-byte size.

6. MPI-2.2 Section 16.1.16 on page 471, line 45.
This is an MPI-2.2 erratum: The constant MPI::_LONG_LONG should be MPI::LONG_LONG.

7. Annex A.1.1 on page 669, Table "Optional datatypes (Fortran)," and MPI-2.2, Annex A.1.1, Table on page 517, lines 34, and 37-41.
This is an MPI-2.2 erratum: The C++ datatype handles MPI::INTEGER16, MPI::REAL16, MPI::F_COMPLEX4, MPI::F_COMPLEX8, MPI::F_COMPLEX16, MPI::F_COMPLEX32 were added to the table.

B.2.2 Changes in MPI-3.0

1. Section 2.6.1 on page 17, Section 16.2 on page 604 and all other chapters.
The C++ bindings were removed from the standard. See errata in Section B.2.1 on page 797 for the latest changes to the MPI C++ binding defined in MPI-2.2.
This change may affect backward compatibility.

2. Section 2.6.1 on page 17, Section 15.1 on page 599 and Section 16.1 on page 603.
The deprecated functions MPI_TYPE_HVECTOR, MPI_TYPE_HINDEXED, MPI_TYPE_STRUCT, MPI_ADDRESS, MPI_TYPE_EXTENT, MPI_TYPE_LB,

MPI_TYPE_UB, MPI_ERRHANDLER_CREATE (and its callback function prototype MPI_Handler_function), MPI_ERRHANDLER_SET, MPI_ERRHANDLER_GET, the deprecated special datatype handles MPI_LB, MPI_UB, and the constants MPI_COMBINER_HINDEXED_INTEGER, MPI_COMBINER_HVECTOR_INTEGER, MPI_COMBINER_STRUCT_INTEGER were removed from the standard.
This change may affect backward compatibility.

3. Section 2.3 on page 10.
Clarified parameter usage for IN parameters. C bindings are now const-correct where backward compatibility is preserved.

4. Section 2.5.4 on page 15 and Section 7.5.4 on page 296.
The recommended C implementation value for MPI_UNWEIGHTED changed from NULL to non-NULL. An additional weight array constant (MPI_WEIGHTS_EMPTY) was introduced.

5. Section 2.5.4 on page 15 and Section 8.1.1 on page 333.
Added the new routine MPI_GET_LIBRARY_VERSION to query library specific versions, and the new constant MPI_MAX_LIBRARY_VERSION_STRING.

6. Sections 2.5.8, 3.2.2, 3.3, 5.9.2, on pages 17, 25, 27, 176, Sections 4.1, 4.1.7, 4.1.8, 4.1.11, 12.3 on pages 83, 106, 108, 111, 482, and Annex A.1.1 on page 669.
New inquiry functions, MPI_TYPE_SIZE_X, MPI_TYPE_GET_EXTENT_X, MPI_TYPE_GET_TRUE_EXTENT_X, and MPI_GET_ELEMENTS_X, return their results as an MPI_Count value, which is a new type large enough to represent element counts in memory, file views, etc. A new function, MPI_STATUS_SET_ELEMENTS_X, modifies the opaque part of an MPI_Status object so that a call to MPI_GET_ELEMENTS_X returns the provided MPI_Count value (in Fortran, `INTEGER (KIND=MPI_COUNT_KIND)`). The corresponding predefined datatype is MPI_COUNT.

7. Chapter 3 on page 23 until Chapter 17 on page 605.
In the C language bindings, the array-arguments' interfaces were modified to consistently use use `[]` instead of `*`.

   Exceptions are MPI_INIT, which continues to use `char ***argv` (correct because of subtle rules regarding the use of the `&` operator with `char *argv[]`), and MPI_INIT_THREAD, which is changed to be consistent with MPI_INIT.

8. Sections 3.2.5, 4.1.5, 4.1.11, 4.2 on pages 30, 101, 111, 131.
The functions MPI_GET_COUNT and MPI_GET_ELEMENTS were defined to set the count argument to MPI_UNDEFINED when that argument would overflow. The functions MPI_PACK_SIZE and MPI_TYPE_SIZE were defined to set the size argument to MPI_UNDEFINED when that argument would overflow. In all other MPI-2.2 routines, the type and semantics of the count arguments remain unchanged, i.e., `int` or `INTEGER`.

9. Section 3.2.6 on page 32, and Section 3.8 on page 64.
MPI_STATUS_IGNORE can be also used in MPI_IPROBE, MPI_PROBE, MPI_IMPROBE, and MPI_MPROBE.

10. Section 3.8 on page 64 and Section 3.11 on page 80.
The use of MPI_PROC_NULL in probe operations was clarified. A special predefined message MPI_MESSAGE_NO_PROC was defined for the use of matching probe (i.e., the new MPI_MPROBE and MPI_IMPROBE) with MPI_PROC_NULL.

11. Sections 3.8.2, 3.8.3, 17.2.4, A.1.1 on pages 67, 69, 654, 669.
Like MPI_PROBE and MPI_IPROBE, the new MPI_MPROBE and MPI_IMPROBE operations allow incoming messages to be queried without actually receiving them, except that MPI_MPROBE and MPI_IMPROBE provide a mechanism to receive the specific message with the new routines MPI_MRECV and MPI_IMRECV regardless of other intervening probe or receive operations. The opaque object MPI_Message, the null handle MPI_MESSAGE_NULL, and the conversion functions MPI_Message_c2f and MPI_Message_f2c were defined.

12. Section 4.1.2 on page 85 and Section 4.1.13 on page 116.
The routine MPI_TYPE_CREATE_HINDEXED_BLOCK and constant MPI_COMBINER_HINDEXED_BLOCK were added.

13. Chapter 5 on page 141 and Section 5.12 on page 196.
Added nonblocking interfaces to all collective operations.

14. Sections 6.4.2, 6.4.4, 11.2.7, on pages 237, 248, 415.
The new routines MPI_COMM_DUP_WITH_INFO, MPI_COMM_SET_INFO, MPI_COMM_GET_INFO, MPI_WIN_SET_INFO, and MPI_WIN_GET_INFO were added. The routine MPI_COMM_DUP must also duplicate info hints.

15. Section 6.4.2 on page 237.
Added MPI_COMM_IDUP.

16. Section 6.4.2 on page 237.
Added the new communicator construction routine MPI_COMM_CREATE_GROUP, which is invoked only by the processes in the group of the new communicator being constructed.

17. Section 6.4.2 on page 237.
Added the MPI_COMM_SPLIT_TYPE routine and the communicator split type constant MPI_COMM_TYPE_SHARED.

18. Section 6.6.2 on page 260.
In MPI-2.2, communication involved in an MPI_INTERCOMM_CREATE operation could interfere with point-to-point communication on the parent communicator with the same tag or MPI_ANY_TAG. This interference has been removed in MPI-3.0.

19. Section 6.8 on page 281.
Section 6.8 on page 238. The constant MPI_MAX_OBJECT_NAME also applies for type and window names.

20. Section 7.5.8 on page 312.
MPI_CART_MAP can also be used for a zero-dimensional topologies.

21. Section 7.6 on page 314 and Section 7.7 on page 323.
The following neighborhood collective communication routines were added to support sparse communication on virtual topology grids: MPI_NEIGHBOR_ALLGATHER, MPI_NEIGHBOR_ALLGATHERV, MPI_NEIGHBOR_ALLTOALL, MPI_NEIGHBOR_ALLTOALLV, MPI_NEIGHBOR_ALLTOALLW and the nonblocking variants MPI_INEIGHBOR_ALLGATHER, MPI_INEIGHBOR_ALLGATHERV, MPI_INEIGHBOR_ALLTOALL, MPI_INEIGHBOR_ALLTOALLV, and MPI_INEIGHBOR_ALLTOALLW. The displacement arguments in MPI_NEIGHBOR_ALLTOALLW and MPI_INEIGHBOR_ALLTOALLW were defined as address size integers. In MPI_DIST_GRAPH_NEIGHBORS, an ordering rule was added for communicators created with MPI_DIST_GRAPH_CREATE_ADJACENT.

22. Section 8.7 on page 355 and Section 12.4.3 on page 487.
The use of MPI_INIT, MPI_INIT_THREAD and MPI_FINALIZE was clarified. After MPI is initialized, the application can access information about the execution environment by querying the new predefined info object MPI_INFO_ENV.

23. Section 8.7 on page 355.
Allow calls to MPI_T routines before MPI_INIT and after MPI_FINALIZE.

24. Chapter 11 on page 401.
Substantial revision of the entire One-sided chapter, with new routines for window creation, additional synchronization methods in passive target communication, new one-sided communication routines, a new memory model, and other changes.

25. Section 14.3 on page 567.
A new MPI Tool Information Interface was added.

The following changes are related to the Fortran language support.

26. Section 2.3 on page 10, and Sections 17.1.1, 17.1.2, 17.1.7 on pages 605, 606, and 621.
The new `mpi_08` Fortran module was introduced.

27. Section 2.5.1 on page 12, and Sections 17.1.2, 17.1.3, 17.1.7 on pages 606, 609, and 621.
Handles to opaque objects were defined as named types within the `mpi_08` Fortran module. The operators `.EQ.`, `.NE.`, `==`, and `/=` were overloaded to allow the comparison of these handles. The handle types and the overloaded operators are also available through the `mpi` Fortran module.

28. Sections 2.5.4, 2.5.5 on pages 15, 16, Sections 17.1.1, 17.1.10, 17.1.11, 17.1.12, 17.1.13 on pages 605, 631, 633, 633, 636, and Sections 17.1.2, 17.1.3, 17.1.7 on pages 606, 609, 621.
Within the `mpi_08` Fortran module, choice buffers were defined as assumed-type and assumed-rank according to Fortran 2008 TS 29113 [41], and the compile-time constant MPI_SUBARRAYS_SUPPORTED was set to `.TRUE.`. With this, Fortran subscript triplets can be used in nonblocking MPI operations; vector subscripts are not supported in nonblocking operations. If the compiler does not support this Fortran TR 29113 feature, the constant is set to `.FALSE.`.

29. Section 2.6.2 on page 19, Section 17.1.2 on page 606, and Section 17.1.7 on page 621.
The ierror dummy arguments are `OPTIONAL` within the `mpi_08` Fortran module.

30. Section 3.2.5 on page 30, Sections 17.1.2, 17.1.3, 17.1.7, on pages 606, 609, 621, and Section 17.2.5 on page 656.
Within the mpi_08 Fortran module, the status was defined as TYPE(MPI_Status). Additionally, within both the mpi and the mpi_f08 modules, the constants MPI_STATUS_SIZE, MPI_SOURCE, MPI_TAG, MPI_ERROR, and TYPE(MPI_Status) are defined. New conversion routines were added: MPI_STATUS_F2F08, MPI_STATUS_F082F, MPI_Status_c2f08, and MPI_Status_f082c, In mpi.h, the new type MPI_F08_status, and the external variables MPI_F08_STATUS_IGNORE and MPI_F08_STATUSES_IGNORE were added.

31. Section 3.6 on page 44.
In Fortran with the mpi module or mpif.h, the type of the buffer_addr argument of MPI_BUFFER_DETACH is incorrectly defined and the argument is therefore unused.

32. Section 4.1 on page 83, Section 4.1.6 on page 104, and Section 17.1.15 on page 637.
The Fortran alignments of basic datatypes within Fortran derived types are implementation dependent; therefore it is recommended to use the BIND(C) attribute for derived types in MPI communication buffers. If an array of structures (in C/C++) or derived types (in Fortran) is to be used in MPI communication buffers, it is recommended that the user creates a portable datatype handle and additionally applies MPI_TYPE_CREATE_RESIZED to this datatype handle.

33. Sections 4.1.10, 5.9.5, 5.9.7, 6.7.4, 6.8, 8.3.1, 8.3.2, 8.3.3, 15.1, 17.1.9 on pages 111, 183, 189, 275, 281, 341, 343, 345, 599, and 623. In some routines, the dummy argument names were changed because they were identical to the Fortran keywords TYPE and FUNCTION. The new dummy argument names must be used because the mpi and mpi_08 modules guarantee keyword-based actual argument lists. The argument name type was changed in MPI_TYPE_DUP, the Fortran USER_FUNCTION of MPI_OP_CREATE, MPI_TYPE_SET_ATTR, MPI_TYPE_GET_ATTR, MPI_TYPE_DELETE_ATTR, MPI_TYPE_SET_NAME, MPI_TYPE_GET_NAME, MPI_TYPE_MATCH_SIZE, the callback prototype definition MPI_Type_delete_attr_function, and the predefined callback function MPI_TYPE_NULL_DELETE_FN; function was changed in MPI_OP_CREATE, MPI_COMM_CREATE_ERRHANDLER, MPI_WIN_CREATE_ERRHANDLER, MPI_FILE_CREATE_ERRHANDLER, and MPI_ERRHANDLER_CREATE. For consistency reasons, INOUBUF was changed to INOUTBUF in MPI_REDUCE_LOCAL, and intracomm to newintracomm in MPI_INTERCOMM_MERGE.

Ticket 0

34. Section 6.7.2 on page 267.
It was clarified that in Fortran, the flag values returned by a comm_copy_attr_fn callback, including MPI_COMM_NULL_COPY_FN and MPI_COMM_DUP_FN, are .FALSE. and .TRUE.; see MPI_COMM_CREATE_KEYVAL.

35. Section 8.2 on page 337.
With the mpi and mpi_f08 Fortran modules, MPI_ALLOC_MEM now also supports TYPE(C_PTR) C-pointers instead of only returning an address-sized integer that may be usable together with a non-standard Cray-pointer.

36. Section 17.1.15 on page 637, and Section 17.1.7 on page 621.
Fortran SEQUENCE and BIND(C) derived application types can now be used as buffers

in MPI operations.

37. Section 17.1.16 on page 639 to Section 17.1.19 on page 648, Section 17.1.7 on page 621, and Section 17.1.8 on page 622.
The sections about Fortran optimization problems and their solutions were partially rewritten and new methods are added, e.g., the use of the ASYNCHRONOUS attribute. The constant MPI_ASYNC_PROTECTS_NONBLOCKING tells whether the semantics of the ASYNCHRONOUS attribute is extended to protect nonblocking operations. The Fortran routine MPI_F_SYNC_REG is added. MPI-3.0 compliance for an MPI library together with a Fortran compiler is defined in Section 17.1.7.

38. Section 17.1.2 on page 606.
Within the mpi_08 Fortran module, dummy arguments are now declared with INTENT=IN, OUT, or INOUT as defined in the mpi_08 interfaces.

39. Section 17.1.3 on page 609, and Section 17.1.7 on page 621.
The existing mpi Fortran module must implement compile-time argument checking.

40. Section 17.1.4 on page 611.
The use of the mpif.h Fortran include file is now strongly discouraged.

41. Section A.1.1, Table "*Predefined functions*" on page 677, Section A.1.3 on page 684, and Section A.3.4 on page 731.
Within the new mpi_f08 module, all callback prototype definitions are now defined with explicit interfaces PROCEDURE(MPI_...) that have the BIND(C) attribute; user-written callbacks must be modified if the mpi_f08 module is used.

42. Section A.1.3 on page 684.
In some routines, the Fortran callback prototype names were changed from ..._FN to ..._FUNCTION to be consistent with the other language bindings.

B.3 Changes from Version 2.1 to Version 2.2

1. Section 2.5.4 on page 15.
It is now guaranteed that predefined named constant handles (as other constants) can be used in initialization expressions or assignments, i.e., also before the call to MPI_INIT.

2. Section 2.6 on page 17, and Section 16.2 on page 604.
The C++ language bindings have been deprecated and may be removed in a future version of the MPI specification.

3. Section 3.2.2 on page 25.
MPI_CHAR for printable characters is now defined for C type char (instead of signed char). This change should not have any impact on applications nor on MPI libraries (except some comment lines), because printable characters could and can be stored in any of the C types char, signed char, and unsigned char, and MPI_CHAR is not allowed for predefined reduction operations.

4. Section 3.2.2 on page 25.
MPI_(U)INT{8,16,32,64}_T, MPI_AINT, MPI_OFFSET, MPI_C_BOOL,

MPI_C_COMPLEX, MPI_C_FLOAT_COMPLEX, MPI_C_DOUBLE_COMPLEX, and MPI_C_LONG_DOUBLE_COMPLEX are now valid predefined MPI datatypes.

5. Section 3.4 on page 37, Section 3.7.2 on page 48, Section 3.9 on page 73, and Section 5.1 on page 141.
   The read access restriction on the send buffer for blocking, non blocking and collective API has been lifted. It is permitted to access for read the send buffer while the operation is in progress.

6. Section 3.7 on page 47.
   The Advice to users for IBSEND and IRSEND was slightly changed.

7. Section 3.7.3 on page 52.
   The advice to free an active request was removed in the Advice to users for MPI_REQUEST_FREE.

8. Section 3.7.6 on page 63.
   MPI_REQUEST_GET_STATUS changed to permit inactive or null requests as input.

9. Section 5.8 on page 168.
   "In place" option is added to MPI_ALLTOALL, MPI_ALLTOALLV, and MPI_ALLTOALLW for intracommunicators.

10. Section 5.9.2 on page 176.
    Predefined parameterized datatypes (e.g., returned by MPI_TYPE_CREATE_F90_REAL) and optional named predefined datatypes (e.g. MPI_REAL8) have been added to the list of valid datatypes in reduction operations.

11. Section 5.9.2 on page 176.
    MPI_(U)INT{8,16,32,64}_T are all considered C integer types for the purposes of the predefined reduction operators. MPI_AINT and MPI_OFFSET are considered Fortran integer types. MPI_C_BOOL is considered a Logical type. MPI_C_COMPLEX, MPI_C_FLOAT_COMPLEX, MPI_C_DOUBLE_COMPLEX, and MPI_C_LONG_DOUBLE_COMPLEX are considered Complex types.

12. Section 5.9.7 on page 189.
    The local routines MPI_REDUCE_LOCAL and MPI_OP_COMMUTATIVE have been added.

13. Section 5.10.1 on page 190.
    The collective function MPI_REDUCE_SCATTER_BLOCK is added to the MPI standard.

14. Section 5.11.2 on page 194.
    Added in place argument to MPI_EXSCAN.

15. Section 6.4.2 on page 237, and Section 6.6 on page 257.
    Implementations that did not implement MPI_COMM_CREATE on intercommunicators will need to add that functionality. As the standard described the behavior of this operation on intercommunicators, it is believed that most implementations already provide this functionality. Note also that the C++ binding for both MPI_COMM_CREATE and MPI_COMM_SPLIT explicitly allow Intercomms.

16. Section 6.4.2 on page 237.
MPI_COMM_CREATE is extended to allow several disjoint subgroups as input if comm is an intracommunicator. If comm is an intercommunicator it was clarified that all processes in the same local group of comm must specify the same value for group.

17. Section 7.5.4 on page 296.
New functions for a scalable distributed graph topology interface has been added. In this section, the functions MPI_DIST_GRAPH_CREATE_ADJACENT and MPI_DIST_GRAPH_CREATE, the constants MPI_UNWEIGHTED, and the derived C++ class Distgraphcomm were added.

18. Section 7.5.5 on page 302.
For the scalable distributed graph topology interface, the functions MPI_DIST_GRAPH_NEIGHBORS_COUNT and MPI_DIST_GRAPH_NEIGHBORS and the constant MPI_DIST_GRAPH were added.

19. Section 7.5.5 on page 302.
Remove ambiguity regarding duplicated neighbors with MPI_GRAPH_NEIGHBORS and MPI_GRAPH_NEIGHBORS_COUNT.

20. Section 8.1.1 on page 333.
The subversion number changed from 1 to 2.

21. Section 8.3 on page 340, Section 15.2 on page 602, and Annex A.1.3 on page 684.
Changed function pointer typedef names MPI_{Comm,File,Win}_errhandler_fn to MPI_{Comm,File,Win}_errhandler_function. Deprecated old "_fn" names.

22. Section 8.7.1 on page 361.
Attribute deletion callbacks on MPI_COMM_SELF are now called in LIFO order. Implementors must now also register all implementation-internal attribute deletion callbacks on MPI_COMM_SELF before returning from MPI_INIT/MPI_INIT_THREAD.

23. Section 11.3.4 on page 423.
The restriction added in MPI 2.1 that the operation MPI_REPLACE in MPI_ACCUMULATE can be used only with predefined datatypes has been removed. MPI_REPLACE can now be used even with derived datatypes, as it was in MPI 2.0. Also, a clarification has been made that MPI_REPLACE can be used only in MPI_ACCUMULATE, not in collective operations that do reductions, such as MPI_REDUCE and others.

24. Section 12.2 on page 475.
Add "*" to the query_fn, free_fn, and cancel_fn arguments to the C++ binding for MPI::Grequest::Start() for consistency with the rest of MPI functions that take function pointer arguments.

25. Section 13.6.2 on page 538, and Table 13.2 on page 540.
MPI_(U)INT{8,16,32,64}_T, MPI_AINT, MPI_OFFSET, MPI_C_COMPLEX, MPI_C_FLOAT_COMPLEX, MPI_C_DOUBLE_COMPLEX, MPI_C_LONG_DOUBLE_COMPLEX, and MPI_C_BOOL are added as predefined datatypes in the external32 representation.

26. Section 17.2.7 on page 661.
The description was modified that it only describes how an MPI implementation behaves, but not how MPI stores attributes internally. The erroneous MPI-2.1 Example 16.17 was replaced with three new examples 17.13, 17.14, and 17.15 on pages 662-663 explicitly detailing cross-language attribute behavior. Implementations that matched the behavior of the old example will need to be updated.

27. Annex A.1.1 on page 669.
Removed type MPI::Fint (compare MPI_Fint in Section A.1.2 on page 682).

28. Annex A.1.1 on page 669. Table *Named Predefined Datatypes*.
Added MPI_(U)INT{8,16,32,64}_T, MPI_AINT, MPI_OFFSET, MPI_C_BOOL, MPI_C_FLOAT_COMPLEX, MPI_C_COMPLEX, MPI_C_DOUBLE_COMPLEX, and MPI_C_LONG_DOUBLE_COMPLEX are added as predefined datatypes.

B.4 Changes from Version 2.0 to Version 2.1

1. Section 3.2.2 on page 25, and Annex A.1 on page 669.
In addition, the MPI_LONG_LONG should be added as an optional type; it is a synonym for MPI_LONG_LONG_INT.

2. Section 3.2.2 on page 25, and Annex A.1 on page 669.
MPI_LONG_LONG_INT, MPI_LONG_LONG (as synonym), MPI_UNSIGNED_LONG_LONG, MPI_SIGNED_CHAR, and MPI_WCHAR are moved from optional to official and they are therefore defined for all three language bindings.

3. Section 3.2.5 on page 30.
MPI_GET_COUNT with zero-length datatypes: The value returned as the count argument of MPI_GET_COUNT for a datatype of length zero where zero bytes have been transferred is zero. If the number of bytes transferred is greater than zero, MPI_UNDEFINED is returned.

4. Section 4.1 on page 83.
General rule about derived datatypes: Most datatype constructors have replication count or block length arguments. Allowed values are non-negative integers. If the value is zero, no elements are generated in the type map and there is no effect on datatype bounds or extent.

5. Section 4.3 on page 138.
MPI_BYTE should be used to send and receive data that is packed using MPI_PACK_EXTERNAL.

6. Section 5.9.6 on page 187.
If comm is an intercommunicator in MPI_ALLREDUCE, then both groups should provide count and datatype arguments that specify the same type signature (i.e., it is not necessary that both groups provide the same count value).

7. Section 6.3.1 on page 228.
MPI_GROUP_TRANSLATE_RANKS and MPI_PROC_NULL: MPI_PROC_NULL is a valid rank for input to MPI_GROUP_TRANSLATE_RANKS, which returns MPI_PROC_NULL as the translated rank.

8. Section 6.7 on page 265.
About the attribute caching functions:

Advice to implementors. High-quality implementations should raise an error when a keyval that was created by a call to MPI_XXX_CREATE_KEYVAL is used with an object of the wrong type with a call to MPI_YYY_GET_ATTR, MPI_YYY_SET_ATTR, MPI_YYY_DELETE_ATTR, or MPI_YYY_FREE_KEYVAL. To do so, it is necessary to maintain, with each keyval, information on the type of the associated user function. (*End of advice to implementors.*)

9. Section 6.8 on page 281.
In MPI_COMM_GET_NAME: In C, a null character is additionally stored at name[resultlen]. resultlen cannot be larger then MPI_MAX_OBJECT_NAME-1. In Fortran, name is padded on the right with blank characters. resultlen cannot be larger then MPI_MAX_OBJECT_NAME.

10. Section 7.4 on page 290.
About MPI_GRAPH_CREATE and MPI_CART_CREATE: All input arguments must have identical values on all processes of the group of comm_old.

11. Section 7.5.1 on page 292.
In MPI_CART_CREATE: If ndims is zero then a zero-dimensional Cartesian topology is created. The call is erroneous if it specifies a grid that is larger than the group size or if ndims is negative.

12. Section 7.5.3 on page 294.
In MPI_GRAPH_CREATE: If the graph is empty, i.e., nnodes == 0, then MPI_COMM_NULL is returned in all processes.

13. Section 7.5.3 on page 294.
In MPI_GRAPH_CREATE: A single process is allowed to be defined multiple times in the list of neighbors of a process (i.e., there may be multiple edges between two processes). A process is also allowed to be a neighbor to itself (i.e., a self loop in the graph). The adjacency matrix is allowed to be non-symmetric.

Advice to users. Performance implications of using multiple edges or a non-symmetric adjacency matrix are not defined. The definition of a node-neighbor edge does not imply a direction of the communication. (*End of advice to users.*)

14. Section 7.5.5 on page 302.
In MPI_CARTDIM_GET and MPI_CART_GET: If comm is associated with a zero-dimensional Cartesian topology, MPI_CARTDIM_GET returns ndims=0 and MPI_CART_GET will keep all output arguments unchanged.

15. Section 7.5.5 on page 302.
In MPI_CART_RANK: If comm is associated with a zero-dimensional Cartesian topology, coord is not significant and 0 is returned in rank.

16. Section 7.5.5 on page 302.
In MPI_CART_COORDS: If comm is associated with a zero-dimensional Cartesian topology, coords will be unchanged.

17. Section 7.5.6 on page 310.
In MPI_CART_SHIFT: It is erroneous to call MPI_CART_SHIFT with a direction that is either negative or greater than or equal to the number of dimensions in the Cartesian communicator. This implies that it is erroneous to call MPI_CART_SHIFT with a comm that is associated with a zero-dimensional Cartesian topology.

18. Section 7.5.7 on page 311.
In MPI_CART_SUB: If all entries in remain_dims are false or comm is already associated with a zero-dimensional Cartesian topology then newcomm is associated with a zero-dimensional Cartesian topology.

18.1. Section 8.1.1 on page 333.
The subversion number changed from 0 to 1.

19. Section 8.1.2 on page 334.
In MPI_GET_PROCESSOR_NAME: In C, a null character is additionally stored at name[resultlen]. resultlen cannot be larger then MPI_MAX_PROCESSOR_NAME-1. In Fortran, name is padded on the right with blank characters. resultlen cannot be larger then MPI_MAX_PROCESSOR_NAME.

20. Section 8.3 on page 340.
MPI_{COMM,WIN,FILE}_GET_ERRHANDLER behave as if a new error handler object is created. That is, once the error handler is no longer needed, MPI_ERRHANDLER_FREE should be called with the error handler returned from MPI_ERRHANDLER_GET or MPI_{COMM,WIN,FILE}_GET_ERRHANDLER to mark the error handler for deallocation. This provides behavior similar to that of MPI_COMM_GROUP and MPI_GROUP_FREE.

21. Section 8.7 on page 355, see explanations to MPI_FINALIZE.
MPI_FINALIZE is collective over all connected processes. If no processes were spawned, accepted or connected then this means over MPI_COMM_WORLD; otherwise it is collective over the union of all processes that have been and continue to be connected, as explained in Section 10.5.4 on page 397.

22. Section 8.7 on page 355.
About MPI_ABORT:

    Advice to users. Whether the errorcode is returned from the executable or from the MPI process startup mechanism (e.g., mpiexec), is an aspect of quality of the MPI library but not mandatory. (*End of advice to users.*)

    Advice to implementors. Where possible, a high-quality implementation will try to return the errorcode from the MPI process startup mechanism (e.g. mpiexec or singleton init). (*End of advice to implementors.*)

23. Section 9 on page 365.
An implementation must support info objects as caches for arbitrary (key, value) pairs, regardless of whether it recognizes the key. Each function that takes hints in the form of an MPI_Info must be prepared to ignore any key it does not recognize. This description of info objects does not attempt to define how a particular function should

react if it recognizes a key but not the associated value. MPI_INFO_GET_NKEYS, MPI_INFO_GET_NTHKEY, MPI_INFO_GET_VALUELEN, and MPI_INFO_GET must retain all (key,value) pairs so that layered functionality can also use the Info object.

24. Section 11.3 on page 417.
MPI_PROC_NULL is a valid target rank in the MPI RMA calls MPI_ACCUMULATE, MPI_GET, and MPI_PUT. The effect is the same as for MPI_PROC_NULL in MPI point-to-point communication. See also item 25 in this list.

25. Section 11.3 on page 417.
After any RMA operation with rank MPI_PROC_NULL, it is still necessary to finish the RMA epoch with the synchronization method that started the epoch. See also item 24 in this list.

26. Section 11.3.4 on page 423.
MPI_REPLACE in MPI_ACCUMULATE, like the other predefined operations, is defined only for the predefined MPI datatypes.

27. Section 13.2.8 on page 500.
About MPI_FILE_SET_VIEW and MPI_FILE_SET_INFO: When an info object that specifies a subset of valid hints is passed to MPI_FILE_SET_VIEW or MPI_FILE_SET_INFO, there will be no effect on previously set or defaulted hints that the info does not specify.

28. Section 13.2.8 on page 500.
About MPI_FILE_GET_INFO: If no hint exists for the file associated with fh, a handle to a newly created info object is returned that contains no key/value pair.

29. Section 13.3 on page 503.
If a file does not have the mode MPI_MODE_SEQUENTIAL, then MPI_DISPLACEMENT_CURRENT is invalid as disp in MPI_FILE_SET_VIEW.

30. Section 13.6.2 on page 538.
The bias of 16 byte doubles was defined with 10383. The correct value is 16383.

31. MPI-2.2, Section 16.1.4 (Section was removed in MPI-3.0).
In the example in this section, the buffer should be declared as `const void* buf`.

32. Section 17.1.9 on page 623.
About MPI_TYPE_CREATE_F90_XXX:

    Advice to implementors. An application may often repeat a call to MPI_TYPE_CREATE_F90_XXX with the same combination of (XXX,p,r). The application is not allowed to free the returned predefined, unnamed datatype handles. To prevent the creation of a potentially huge amount of handles, the MPI implementation should return the same datatype handle for the same (REAL/COMPLEX/INTEGER,p,r) combination. Checking for the combination (p,r) in the preceding call to MPI_TYPE_CREATE_F90_XXX and using a hash-table to find formerly generated handles should limit the overhead of finding a previously generated datatype with same combination of (XXX,p,r). (*End of advice to implementors.*)

33. Section A.1.1 on page 669.
    MPI_BOTTOM is defined as `void * const MPI::BOTTOM`.