

this routine with predefined datatypes employed by the user.

Datarep Conversion Functions

```

typedef int MPI_Datarep_conversion_function(void *userbuf,
      MPI_Datatype datatype, int count, void *filebuf,
      MPI_Offset position, void *extra_state);

ABSTRACT INTERFACE
  SUBROUTINE MPI_Datarep_conversion_function(userbuf, datatype, count,
    filebuf, position, extra_state, ierror)
    USE, INTRINSIC :: ISO_C_BINDING, ONLY : C_PTR
    TYPE(C_PTR), VALUE :: userbuf, filebuf
    TYPE(MPI_Datatype) :: datatype
    INTEGER :: count, ierror
    INTEGER(KIND=MPI_OFFSET_KIND) :: position
    INTEGER(KIND=MPI_ADDRESS_KIND) :: extra_state

  SUBROUTINE DATAREP_CONVERSION_FUNCTION(USERBUF, DATATYPE, COUNT, FILEBUF,
    POSITION, EXTRA_STATE, IERROR)
    <TYPE> USERBUF(*), FILEBUF(*)
    INTEGER COUNT, DATATYPE, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) POSITION
    INTEGER(KIND=MPI_ADDRESS_KIND) EXTRA_STATE

```

The function `read_conversion_fn` must convert from file data representation to native representation. Before calling this routine, MPI allocates and fills `filebuf` with `count` contiguous data items. The type of each data item matches the corresponding entry for the predefined datatype in the type signature of `datatype`. The function is passed, in `extra_state`, the argument that was passed to the `MPI_REGISTER_DATAREP` call. The function must copy all `count` data items from `filebuf` to `userbuf` in the distribution described by `datatype`, converting each data item from file representation to native representation. `datatype` will be equivalent to the datatype that the user passed to the read function. If the size of `datatype` is less than the size of the `count` data items, the conversion function must treat `datatype` as being contiguously tiled over the `userbuf`. The conversion function must begin storing converted data at the location in `userbuf` specified by `position` into the (tiled) `datatype`.

Advice to users. Although the conversion functions have similarities to `MPI_PACK` and `MPI_UNPACK`, one should note the differences in the use of the arguments `count` and `position`. In the conversion functions, `count` is a count of data items (i.e., count of typemap entries of `datatype`), and `position` is an index into this typemap. In `MPI_PACK`, `incount` refers to the number of whole datatypes, and `position` is a number of bytes. (*End of advice to users.*)

Advice to implementors. A converted read operation could be implemented as follows:

1. Get file extent of all data items
2. Allocate a filebuf large enough to hold all count data items
3. Read data from file into filebuf

4. Call `read_conversion_fn` to convert data and place it into `userbuf`
5. Deallocate `filebuf`

(End of advice to implementors.)

If MPI cannot allocate a buffer large enough to hold all the data to be converted from a read operation, it may call the conversion function repeatedly using the same `datatype` and `userbuf`, and reading successive chunks of data to be converted in `filebuf`. For the first call (and in the case when all the data to be converted fits into `filebuf`), MPI will call the function with `position` set to zero. Data converted during this call will be stored in the `userbuf` according to the first `count` data items in `datatype`. Then in subsequent calls to the conversion function, MPI will increment the value in `position` by the `count` of items converted in the previous call, and the `userbuf` pointer will be unchanged.

Rationale. Passing the conversion function a position and one datatype for the transfer allows the conversion function to decode the datatype only once and cache an internal representation of it on the datatype. Then on subsequent calls, the conversion function can use the `position` to quickly find its place in the datatype and continue storing converted data where it left off at the end of the previous call. *(End of rationale.)*

Advice to users. Although the conversion function may usefully cache an internal representation on the datatype, it should not cache any state information specific to an ongoing conversion operation, since it is possible for the same datatype to be used concurrently in multiple conversion operations. *(End of advice to users.)*

The function `write_conversion_fn` must convert from native representation to file data representation. Before calling this routine, MPI allocates `filebuf` of a size large enough to hold `count` contiguous data items. The type of each data item matches the corresponding entry for the predefined datatype in the type signature of `datatype`. The function must copy `count` data items from `userbuf` in the distribution described by `datatype`, to a contiguous distribution in `filebuf`, converting each data item from native representation to file representation. If the size of `datatype` is less than the size of `count` data items, the conversion function must treat `datatype` as being contiguously tiled over the `userbuf`.

The function must begin copying at the location in `userbuf` specified by `position` into the (tiled) `datatype`. `datatype` will be equivalent to the datatype that the user passed to the write function. The function is passed, in `extra_state`, the argument that was passed to the `MPI_REGISTER_DATAREP` call.

The predefined constant `MPI_CONVERSION_FN_NULL` may be used as either `write_conversion_fn` or `read_conversion_fn`. In that case, MPI will not attempt to invoke `write_conversion_fn` or `read_conversion_fn`, respectively, but will perform the requested data access using the native data representation.

An MPI implementation must ensure that all data accessed is converted, either by using a `filebuf` large enough to hold all the requested data items or else by making repeated calls to the conversion function with the same `datatype` argument and appropriate values for `position`.

An implementation will only invoke the callback routines in this section (`read_conversion_fn`, `write_conversion_fn`, and `dtype_file_extent_fn`) when one of the read or

Topologies

C type: <code>const int</code> (or unnamed <code>enum</code>)
Fortran type: <code>INTEGER</code>
<code>MPI_GRAPH</code>
<code>MPI_CART</code>
<code>MPI_DIST_GRAPH</code>

Predefined functions

C/Fortran name
C type
/ Fortran type with <code>mpi</code> module / Fortran type with <code>mpi_f08</code> module
<code>MPI_COMM_NULL_COPY_FN</code>
<code>MPI_Comm_copy_attr_function</code>
/ <code>COMM_COPY_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Comm_copy_attr_function)</code> ¹⁾
<code>MPI_COMM_DUP_FN</code>
<code>MPI_Comm_copy_attr_function</code>
/ <code>COMM_COPY_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Comm_copy_attr_function)</code> ¹⁾
<code>MPI_COMM_NULL_DELETE_FN</code>
<code>MPI_Comm_delete_attr_function</code>
/ <code>COMM_DELETE_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Comm_delete_attr_function)</code> ¹⁾
<code>MPI_WIN_NULL_COPY_FN</code>
<code>MPI_Win_copy_attr_function</code>
/ <code>WIN_COPY_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Win_copy_attr_function)</code> ¹⁾
<code>MPI_WIN_DUP_FN</code>
<code>MPI_Win_copy_attr_function</code>
/ <code>WIN_COPY_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Win_copy_attr_function)</code> ¹⁾
<code>MPI_WIN_NULL_DELETE_FN</code>
<code>MPI_Win_delete_attr_function</code>
/ <code>WIN_DELETE_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Win_delete_attr_function)</code> ¹⁾
<code>MPI_TYPE_NULL_COPY_FN</code>
<code>MPI_Type_copy_attr_function</code>
/ <code>TYPE_COPY_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Type_copy_attr_function)</code> ¹⁾
<code>MPI_TYPE_DUP_FN</code>
<code>MPI_Type_copy_attr_function</code>
/ <code>TYPE_COPY_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Type_copy_attr_function)</code> ¹⁾
<code>MPI_TYPE_NULL_DELETE_FN</code>
<code>MPI_Type_delete_attr_function</code>
/ <code>TYPE_DELETE_ATTR_FUNCTION</code> / <code>PROCEDURE(MPI_Type_delete_attr_function)</code> ¹⁾
<code>MPI_CONVERSION_FN_NULL</code>
<code>MPI_Datarep_conversion_function</code>
/ <code>DATAREP_CONVERSION_FUNCTION</code> / <code>PROCEDURE(MPI_Datarep_conversion_function)</code> ¹⁾

¹ See the advice to implementors (on page 270) and advice to users (on page 270) on the predefined Fortran functions `MPI_COMM_NULL_COPY_FN`, ... in Section 6.7.2.

```

int MPI_Win_free(MPI_Win *win)
int MPI_Win_get_group(MPI_Win win, MPI_Group *group)
int MPI_Win_get_info(MPI_Win win, MPI_Info *info_used)
int MPI_Win_lock_all(int assert, MPI_Win win)
int MPI_Win_lock(int lock_type, int rank, int assert, MPI_Win win)
int MPI_Win_post(MPI_Group group, int assert, MPI_Win win)
int MPI_Win_set_info(MPI_Win win, MPI_Info info)
int MPI_Win_shared_query(MPI_Win win, int rank, MPI_Aint *size,
    int *disp_unit, void *baseptr)
int MPI_Win_start(MPI_Group group, int assert, MPI_Win win)
int MPI_Win_sync(MPI_Win win)
int MPI_Win_test(MPI_Win win, int *flag)
int MPI_Win_unlock_all(MPI_Win win)
int MPI_Win_unlock(int rank, MPI_Win win)
int MPI_Win_wait(MPI_Win win)

```

A.2.10 External Interfaces C Bindings

```

int MPI_Grequest_complete(MPI_Request request)
int MPI_Grequest_start(MPI_Grequest_query_function *query_fn,
    MPI_Grequest_free_function *free_fn,
    MPI_Grequest_cancel_function *cancel_fn, void *extra_state,
    MPI_Request *request)
int MPI_Init_thread(int *argc, char ***argv, int required, int *provided)
int MPI_Is_thread_main(int *flag)
int MPI_Query_thread(int *provided)
int MPI_Status_set_cancelled(MPI_Status *status, int flag)
int MPI_Status_set_elements(MPI_Status *status, MPI_Datatype datatype,
    int count)
int MPI_Status_set_elements_x(MPI_Status *status, MPI_Datatype datatype,
    MPI_Count count)

```

A.2.11 I/O C Bindings

```

int MPI_CONVERSION_FN_NULL(void *userbuf, MPI_Datatype datatype, int count,
    void *filebuf, MPI_Offset position, void *extra_state)

```

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

```

1      INTEGER, INTENT(IN) :: count
2      INTEGER, OPTIONAL, INTENT(OUT) :: ierror
3
4      MPI_Status_set_elements_x(status, datatype, count, ierror)
5      TYPE(MPI_Status), INTENT(INOUT) :: status
6      TYPE(MPI_Datatype), INTENT(IN) :: datatype
7      INTEGER(KIND = MPI_COUNT_KIND), INTENT(IN) :: count
8      INTEGER, OPTIONAL, INTENT(OUT) :: ierror
9

```

A.3.11 I/O Fortran 2008 Bindings

```

12      MPI_CONVERSION_FN_NULL(userbuf, datatype, count, filebuf, position,
13          extra_state, ierror)
14      USE, INTRINSIC :: ISO_C_BINDING, ONLY : C_PTR
15      TYPE(C_PTR), VALUE :: userbuf, filebuf
16      TYPE(MPI_Datatype) :: datatype
17      INTEGER :: count, ierror
18      INTEGER(KIND=MPI_OFFSET_KIND) :: position
19      INTEGER(KIND=MPI_ADDRESS_KIND) :: extra_state

```

#389

```

20      MPI_File_close(fh, ierror)
21      TYPE(MPI_File), INTENT(INOUT) :: fh
22      INTEGER, OPTIONAL, INTENT(OUT) :: ierror
23
24      MPI_File_delete(filename, info, ierror)
25      CHARACTER(LEN=*), INTENT(IN) :: filename
26      TYPE(MPI_Info), INTENT(IN) :: info
27      INTEGER, OPTIONAL, INTENT(OUT) :: ierror
28
29      MPI_File_get_amode(fh, amode, ierror)
30      TYPE(MPI_File), INTENT(IN) :: fh
31      INTEGER, INTENT(OUT) :: amode
32      INTEGER, OPTIONAL, INTENT(OUT) :: ierror
33
34      MPI_File_get_atomicity(fh, flag, ierror)
35      TYPE(MPI_File), INTENT(IN) :: fh
36      LOGICAL, INTENT(OUT) :: flag
37      INTEGER, OPTIONAL, INTENT(OUT) :: ierror
38
39      MPI_File_get_byte_offset(fh, offset, disp, ierror)
40      TYPE(MPI_File), INTENT(IN) :: fh
41      INTEGER(KIND=MPI_OFFSET_KIND), INTENT(IN) :: offset
42      INTEGER(KIND=MPI_OFFSET_KIND), INTENT(OUT) :: disp
43      INTEGER, OPTIONAL, INTENT(OUT) :: ierror
44
45      MPI_File_get_group(fh, group, ierror)
46      TYPE(MPI_File), INTENT(IN) :: fh
47      TYPE(MPI_Group), INTENT(OUT) :: group
48      INTEGER, OPTIONAL, INTENT(OUT) :: ierror
49
50      MPI_File_get_info(fh, info_used, ierror)

```

```

    INTEGER REQUEST, IERROR
    EXTERNAL QUERY_FN, FREE_FN, CANCEL_FN
    INTEGER (KIND=MPI_ADDRESS_KIND) EXTRA_STATE
MPI_INIT_THREAD(REQUIRED, PROVIDED, IERROR)
    INTEGER REQUIRED, PROVIDED, IERROR
MPI_IS_THREAD_MAIN(FLAG, IERROR)
    LOGICAL FLAG
    INTEGER IERROR
MPI_QUERY_THREAD(PROVIDED, IERROR)
    INTEGER PROVIDED, IERROR
MPI_STATUS_SET_CANCELLED(STATUS, FLAG, IERROR)
    INTEGER STATUS(MPI_STATUS_SIZE), IERROR
    LOGICAL FLAG
MPI_STATUS_SET_ELEMENTS(STATUS, DATATYPE, COUNT, IERROR)
    INTEGER STATUS(MPI_STATUS_SIZE), DATATYPE, COUNT, IERROR
MPI_STATUS_SET_ELEMENTS_X(STATUS, DATATYPE, COUNT, IERROR)
    INTEGER STATUS(MPI_STATUS_SIZE), DATATYPE, IERROR
    INTEGER (KIND=MPI_COUNT_KIND) COUNT

```

A.4.11 I/O Fortran Bindings

```

MPI_CONVERSION_FN_NULL(USERBUF, DATATYPE, COUNT, FILEBUF, POSITION,
    EXTRA_STATE, IERROR)
    <TYPE> USERBUF(*), FILEBUF(*)
    INTEGER COUNT, DATATYPE, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) POSITION
    INTEGER(KIND=MPI_ADDRESS_KIND) EXTRA_STATE
MPI_FILE_CLOSE(FH, IERROR)
    INTEGER FH, IERROR
MPI_FILE_DELETE(FILENAME, INFO, IERROR)
    CHARACTER*(*) FILENAME
    INTEGER INFO, IERROR
MPI_FILE_GET_AMODE(FH, AMODE, IERROR)
    INTEGER FH, AMODE, IERROR
MPI_FILE_GET_ATOMICITY(FH, FLAG, IERROR)
    INTEGER FH, IERROR
    LOGICAL FLAG
MPI_FILE_GET_BYTE_OFFSET(FH, OFFSET, DISP, IERROR)
    INTEGER FH, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) OFFSET, DISP
MPI_FILE_GET_GROUP(FH, GROUP, IERROR)

```

#389

Annex B

Change-Log

This annex 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.

B.1 Changes from Version 3.0 to Version 3.1

B.1.1 Fixes to Errata in Previous Versions of MPI

1. Sections [2.6.4](#) and [4.1.5](#) on pages [20](#) and [101](#).
MPI-3.0 Sections 2.6.4 and 4.1.5 on pages 19 and 102.
The use of the intrinsic operators "+" and "-" for absolute addresses is substituted by MPI_AINT_ADD and MPI_AINT_DIFF. InC, they can be implemented as macros.
2. Section [7.6](#) on page [314](#).
MPI-3.0 Sections 7.6, on pages 314.
In the case of virtual general graph topologies (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).
3. Section [11.2.2](#) on page [405](#).
MPI-3.0 Section 11.2.2 page 407.
The same_size info key can be used with all window flavors.
4. Section [11.3.4](#) on page [423](#).
MPI-3.0 Section 11.3.4 page 424.
Origin buffer arguments to MPI_GET_ACCUMULATE are ignored when the MPI_NO_OP operation is used.
5. Section [11.3.4](#) on page [423](#).
MPI-3.0 Section 11.3.4 page 424.
Clarify the roles of origin, result, and target communication parameters in MPI_GET_ACCUMULATE.

6. Annexes A.2, A.3, and A.4 on pages 690, 712, and 761.
MPI-3.0 Annexes A.2, A.3, and A.4.
The predefined callback `MPI_CONVERSION_FN_NULL` was added to all three annexes.
7. Section ?? on page ??.
MPI-3.0 Section xxx page nnn.
xxx

B.1.2 Changes in MPI-3.0

1. Section ?? on page ??.
xxx

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, 538, and
667, 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” reduc-
tion 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 538, and
MPI-2.2, Section 13.5.3, Table 13.2 on page 433.

1 MPI_COMBINER_SUBARRAY, 117, 121, 677
 2 MPI_COMBINER_VECTOR, 117, 120, 677
 3 MPI_COMM_DUP_FN, 18, 269, 675, 799
 4 MPI_COMM_NULL, 227, 240, 241, 243–245,
 5 247, 248, 283, 292, 294, 379, 398–400,
 6 674, 803
 7 MPI_COMM_NULL_COPY_FN, 18, 269, 606,
 8 658, 675, 799
 9 MPI_COMM_NULL_DELETE_FN, 18, 269,
 10 675
 11 MPI_COMM_PARENT, 283
 12 MPI_COMM_SELF, 227, 243, 266, 283, 361,
 13 398, 492, 673, 801
 14 MPI_COMM_TYPE_SHARED, 248, 673, 797
 15 MPI_COMM_WORLD, 15, 22, 27, 28, 227,
 16 229, 236, 237, 252, 261, 283, 293,
 17 334–336, 340, 342, 351, 357, 359, 360,
 18 362, 371, 372, 374, 375, 379, 381,
 19 395–398, 486, 532, 553, 576, 584, 652,
 20 663, 673, 805
 21 MPI_COMPLEX, 25, 177, 536, 622, 672
 22 MPI_COMPLEX16, 177, 672
 23 MPI_COMPLEX32, 177, 672
 24 MPI_COMPLEX4, 177, 672
 25 MPI_COMPLEX8, 177, 672
 26 MPI_CONGRUENT, 237, 259, 673
 27 MPI_CONVERSION_FN_NULL, 541, 675
 28 MPI_COUNT, 25, 27, 177, 569, 671, 672, 796
 29 MPI_COUNT_KIND, 15, 26, 670
 30 MPI_CXX_BOOL, 27, 177, 672, 794
 31 MPI_CXX_DOUBLE_COMPLEX, 27, 177,
 32 672, 794
 33 MPI_CXX_FLOAT_COMPLEX, 27, 177, 672,
 34 794
 35 MPI_CXX_LONG_DOUBLE_COMPLEX, 27,
 36 177, 672, 794
 37 MPI_DATATYPE_NULL, 111, 674
 38 MPI_DISPLACEMENT_CURRENT, 502,
 39 678, 806
 40 MPI_DIST_GRAPH, 302, 675, 801
 41 MPI_DISTRIBUTE_BLOCK, 98, 678
 42 MPI_DISTRIBUTE_CYCLIC, 98, 678
 43 MPI_DISTRIBUTE_DFLT_DARG, 98, 678
 44 MPI_DISTRIBUTE_NONE, 98, 678
 45 MPI_DOUBLE, 26, 176, 569, 578–580, 621,
 46 671
 47 MPI_DOUBLE_COMPLEX, 25, 177, 536, 622,
 48 672
 MPI_DOUBLE_INT, 180, 673
 MPI_DOUBLE_PRECISION, 25, 176, 622,
 672
 MPI_DUP_FN, 18, 269, 598, 676
 MPI_ERR_ACCESS, 349, 495, 554, 668
 MPI_ERR_AMODE, 349, 493, 554, 668

MPI_ERR_ARG, 348, 667
 MPI_ERR_ASSERT, 348, 452, 668
 MPI_ERR_BAD_FILE, 349, 554, 668
 MPI_ERR_BASE, 338, 348, 452, 668
 MPI_ERR_BUFFER, 348, 667
 MPI_ERR_COMM, 348, 667
 MPI_ERR_CONVERSION, 349, 542, 554, 668
 MPI_ERR_COUNT, 348, 667
 MPI_ERR_DIMS, 348, 667
 MPI_ERR_DISP, 348, 452, 668
 MPI_ERR_DUP_DATAREP, 349, 539, 554,
 668
 MPI_ERR_FILE, 349, 554, 668
 MPI_ERR_FILE_EXISTS, 349, 554, 668
 MPI_ERR_FILE_IN_USE, 349, 495, 554, 668
 MPI_ERR_GROUP, 348, 667
 MPI_ERR_IN_STATUS, 30, 32, 53, 59, 61,
 342, 348, 477, 507, 668
 MPI_ERR_INFO, 348, 668
 MPI_ERR_INFO_KEY, 348, 366, 668
 MPI_ERR_INFO_NOKEY, 348, 367, 668
 MPI_ERR_INFO_VALUE, 348, 366, 668
 MPI_ERR_INTERN, 348, 667
 MPI_ERR_IO, 349, 554, 668
 MPI_ERR_KEYVAL, 279, 348, 668
 MPI_ERR_LASTCODE, 347, 349, 351, 352,
 594, 669
 MPI_ERR_LOCKTYPE, 348, 452, 668
 MPI_ERR_NAME, 348, 392, 668
 MPI_ERR_NO_MEM, 338, 348, 668
 MPI_ERR_NO_SPACE, 349, 554, 668
 MPI_ERR_NO_SUCH_FILE, 349, 494, 554,
 668
 MPI_ERR_NOT_SAME, 349, 554, 668
 MPI_ERR_OP, 348, 452, 667
 MPI_ERR_OTHER, 347, 348, 667
 MPI_ERR_PENDING, 59, 348, 667
 MPI_ERR_PORT, 348, 389, 668
 MPI_ERR_QUOTA, 349, 554, 668
 MPI_ERR_RANK, 348, 452, 667
 MPI_ERR_READ_ONLY, 349, 554, 668
 MPI_ERR_REQUEST, 348, 667
 MPI_ERR_RMA_ATTACH, 349, 452, 668
 MPI_ERR_RMA_CONFLICT, 348, 452, 668
 MPI_ERR_RMA_FLAVOR, 349, 409, 452, 668
 MPI_ERR_RMA_RANGE, 349, 452, 668
 MPI_ERR_RMA_SHARED, 349, 452, 668
 MPI_ERR_RMA_SYNC, 348, 452, 668
 MPI_ERR_ROOT, 348, 667
 MPI_ERR_SERVICE, 348, 391, 668
 MPI_ERR_SIZE, 348, 452, 668
 MPI_ERR_SPAWN, 348, 377, 378, 668
 MPI_ERR_TAG, 348, 667
 MPI_ERR_TOPOLOGY, 348, 667

- MPI_COMM_FREE_KEYVAL, 18, 266, [270](#),
[279](#), 598
- MPI_COMM_GET_ATTR, 18, 266, [271](#), [271](#),
[279](#), 334, 599, 612, 659, 660, 662
- MPI_COMM_GET_ERRHANDLER, 18, 341,
[343](#), 601, 804
- MPI_COMM_GET_INFO, [249](#), 250, 796
- MPI_COMM_GET_NAME, 282, [282](#), 283, 803
- MPI_COMM_GET_PARENT, 283, 375, 378,
[378](#), 379
- MPI_COMM_GROUP, 14, 227, 230, [230](#), 235,
[236](#), 259, 341, 804
- MPI_COMM_IDUP, 235, 237, 239, [239](#), 248,
[249](#), 257, 266, 269, 272, 279, 796
- MPI_COMM_JOIN, 399, [399](#), 400
- MPI_COMM_NULL_COPY_FN, 18, 269, [269](#),
[270](#), 606, 658, 675, 799
- MPI_COMM_NULL_DELETE_FN, 18, 269,
[269](#), 270, 675
- MPI_COMM_RANK, 236, [236](#), 259, 613
- MPI_COMM_RANK_F08, 613
- MPI_COMM_REMOTE_GROUP, [260](#)
- MPI_COMM_REMOTE_SIZE, 260, [260](#)
- MPI_COMM_SET_ATTR, 18, 266, 269, [270](#),
[279](#), 598, 612, 659, 660, 663
- MPI_COMM_SET_ERRHANDLER, 18, 341,
[342](#), 601
- MPI_COMM_SET_INFO, 248, [249](#), [249](#), 796
- MPI_COMM_SET_NAME, 281, [281](#), 282
- MPI_COMM_SIZE, [235](#), 236, 259
- MPI_COMM_SPAWN, 356, 362, 363, 372–374,
[374](#), 375, 377–379, 381–383, 395–397
- MPI_COMM_SPAWN_MULTIPLE, 356, 363,
[372](#), 373, 378, [380](#), 381, 396, 397
- MPI_COMM_SPLIT, 237, 240, 241, 244, [244](#),
[245](#), 246, 286, 291, 292, 294, 312–314,
801
- MPI_COMM_SPLIT_TYPE, [247](#), 248, 797
- MPI_COMM_TEST_INTER, 258, [259](#)
- MPI_COMM_WORLD, 487
- MPI_COMPARE_AND_SWAP, 401, 417, [429](#),
467
- [MPI_CONVERSION_FN_NULL](#), [541](#), 675,
[794](#)
- MPI_CWIN_GET_ATTR, 612
- MPI_DIMS_CREATE, 291, 293, [293](#)
- MPI_DIST_GRAPH_CREATE, 248, 290, 291,
[296](#), [298](#), 299, 301, 309, 310, 314, 801
- MPI_DIST_GRAPH_CREATE_ADJACENT,
[248](#), 290, 291, [296](#), [296](#), 297, 301, 309,
[314](#), 797, 801
- MPI_DIST_GRAPH_NEIGHBOR_COUNT,
[310](#)
- MPI_DIST_GRAPH_NEIGHBORS, 291, 308,
[309](#), [309](#), 314, 797, 801
- MPI_DIST_GRAPH_NEIGHBORS_COUNT,
[291](#), 308, [308](#), 309, 794, 801
- MPI_DUP_FN, 18, 269, [598](#), 676
- MPI_ERRHANDLER_C2F, [653](#)
- MPI_ERRHANDLER_CREATE, 18, 601, 795,
799
- MPI_ERRHANDLER_F2C, [653](#)
- MPI_ERRHANDLER_FREE, 341, [346](#), 357,
804
- MPI_ERRHANDLER_GET, 18, 601, 795, 804
- MPI_ERRHANDLER_SET, 18, 601, 795
- MPI_ERROR_CLASS, 347, 350, [350](#), 594
- MPI_ERROR_STRING, 347, [347](#), 350, 352
- MPI_EXSCAN, 142, 145, 176, 183, 194, [194](#),
[214](#), 801
- MPI_F_SYNC_REG, 102, 604, 620, [620](#), 621,
[640](#), 642, 643, 645, 799
- MPI_FETCH_AND_OP, 401, 417, 425, 427,
[428](#), [428](#)
- MPI_FILE_C2F, [653](#)
- MPI_FILE_CALL_ERRHANDLER, [353](#), 354
- MPI_FILE_CLOSE, 398, 491, 492, [493](#), 494
- MPI_FILE_CREATE_ERRHANDLER, 341,
[345](#), 346, 684, 686, 799
- MPI_FILE_DELETE, 493, 494, [494](#), 498, 501,
553
- MPI_FILE_F2C, [653](#)
- MPI_FILE_GET_AMODE, 497, [497](#)
- MPI_FILE_GET_ATOMICITY, 545, [545](#)
- MPI_FILE_GET_BYTE_OFFSET, 512, 519,
[519](#), 520, 525
- MPI_FILE_GET_ERRHANDLER, 341, [346](#),
553, 804
- MPI_FILE_GET_GROUP, 497, [497](#)
- MPI_FILE_GET_INFO, 499, [499](#), 501, 805
- MPI_FILE_GET_POSITION, 519, [519](#)
- MPI_FILE_GET_POSITION_SHARED, 524,
[525](#), [525](#), 545
- MPI_FILE_GET_SIZE, [496](#), 497, 548
- MPI_FILE_GET_TYPE_EXTENT, 535, [536](#),
542
- MPI_FILE_GET_VIEW, 504, [504](#)
- MPI_FILE_IXXX, 506
- MPI_FILE_IREAD, 505, 516, [516](#), 526, 543
- MPI_FILE_IREAD_ALL, 505, 517, [517](#)
- MPI_FILE_IREAD_AT, 505, 510, [510](#)
- MPI_FILE_IREAD_AT_ALL, 505, [510](#), 511
- MPI_FILE_IREAD_SHARED, 505, [522](#), [522](#)
- MPI_FILE_IWRITE, 505, [517](#), 518
- MPI_FILE_IWRITE_ALL, 505, 518, [518](#)
- MPI_FILE_IWRITE_AT, 505, 511, [511](#)
- MPI_FILE_IWRITE_AT_ALL, 505, [512](#), [512](#)