

```

IERROR
1
2
MPI_NEIGHBOR_ALLTOALL(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, REVCOUNT,
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

```

A.4.6 MPI Environmental Management Fortran Bindings

```

DOUBLE PRECISION MPI_WTICK()
DOUBLE PRECISION MPI_WTIME()
MPI_ABORT(COMM, ERRORCODE, IERROR)
MPI_ADD_ERROR_CLASS(ERRORCLASS, IERROR)
MPI_ADD_ERROR_CODE(ERRORCLASS, ERRORCODE, IERROR)
MPI_ADD_ERROR_STRING(ERRORCODE, STRING, IERROR)
MPI_ALLOC_MEM(SIZE, INFO, BASEPTR, IERROR)
If the Fortran compiler provides TYPE(C_PTR), then overloaded by:
INTERFACE MPI_ALLOC_MEM
SUBROUTINE MPI_ALLOC_MEM(SIZE, INFO, BASEPTR, IERROR)
IMPORT :: MPI_ADDRESS_KIND
INTEGER :: INFO, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE, BASEPTR

```

```

1  |      END SUBROUTINE
2      SUBROUTINE MPI_ALLOC_MEM_CPTR(SIZE, INFO, BASEPTR, IERROR)
3          USE, INTRINSIC :: ISO_C_BINDING, ONLY : C_PTR
4  |      IMPORT :: MPI_ADDRESS_KIND
5          INTEGER :: INFO, IERROR
6          INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE
7          TYPE(C_PTR) :: BASEPTR
8      END SUBROUTINE
9      END INTERFACE
10
11     MPI_COMM_CALL_ERRHANDLER(COMM, ERRORCODE, IERROR)
12         INTEGER COMM, ERRORCODE, IERROR
13
14     MPI_COMM_CREATE_ERRHANDLER(COMM_ERRHANDLER_FN, ERRHANDLER, IERROR)
15         EXTERNAL COMM_ERRHANDLER_FN
16         INTEGER ERRHANDLER, IERROR
17
18     MPI_COMM_GET_ERRHANDLER(COMM, ERRHANDLER, IERROR)
19         INTEGER COMM, ERRHANDLER, IERROR
20
21     MPI_COMM_SET_ERRHANDLER(COMM, ERRHANDLER, IERROR)
22         INTEGER COMM, ERRHANDLER, IERROR
23
24     MPI_ERRHANDLER_FREE(ERRHANDLER, IERROR)
25         INTEGER ERRHANDLER, IERROR
26
27     MPI_ERROR_CLASS(ERRORCODE, ERRORCLASS, IERROR)
28         INTEGER ERRORCODE, ERRORCLASS, IERROR
29
30     MPI_ERROR_STRING(ERRORCODE, STRING, RESULTLEN, IERROR)
31         INTEGER ERRORCODE, RESULTLEN, IERROR
32         CHARACTER*(*) STRING
33
34     MPI_FILE_CALL_ERRHANDLER(FH, ERRORCODE, IERROR)
35         INTEGER FH, ERRORCODE, IERROR
36
37     MPI_FILE_CREATE_ERRHANDLER(FILE_ERRHANDLER_FN, ERRHANDLER, IERROR)
38         EXTERNAL FILE_ERRHANDLER_FN
39         INTEGER ERRHANDLER, IERROR
40
41     MPI_FILE_GET_ERRHANDLER(FILE, ERRHANDLER, IERROR)
42         INTEGER FILE, ERRHANDLER, IERROR
43
44     MPI_FILE_SET_ERRHANDLER(FILE, ERRHANDLER, IERROR)
45         INTEGER FILE, ERRHANDLER, IERROR
46
47     MPI_FINALIZED(FLAG, IERROR)
48         LOGICAL FLAG
49         INTEGER IERROR
50
51     MPI_FINALIZE(IERROR)
52         INTEGER IERROR
53
54     MPI_FREE_MEM(BASE, IERROR)

```

```

1  MPI_PUT(ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK,
2          TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, WIN, IERROR)
3      <type> ORIGIN_ADDR(*)
4      INTEGER(KIND=MPI_ADDRESS_KIND) TARGET_DISP
5      INTEGER ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK, TARGET_COUNT,
6      TARGET_DATATYPE, WIN, IERROR
7
8  MPI_RACCUMULATE(ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK,
9                  TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, OP, WIN, REQUEST,
10                 IERROR)
11      <type> ORIGIN_ADDR(*)
12      INTEGER(KIND=MPI_ADDRESS_KIND) TARGET_DISP
13      INTEGER ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK, TARGET_COUNT,
14      TARGET_DATATYPE, OP, WIN, REQUEST, IERROR
15
16  MPI_RGET_ACCUMULATE(ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE,
17                     RESULT_ADDR, RESULT_COUNT, RESULT_DATATYPE, TARGET_RANK,
18                     TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, OP, WIN, REQUEST,
19                     IERROR)
20      <type> ORIGIN_ADDR(*), RESULT_ADDR(*)
21      INTEGER(KIND=MPI_ADDRESS_KIND) TARGET_DISP
22      INTEGER ORIGIN_COUNT, ORIGIN_DATATYPE, RESULT_COUNT, RESULT_DATATYPE,
23      TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, OP, WIN, REQUEST, IERROR
24
25  MPI_RGET(ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK,
26           TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, WIN, REQUEST,
27           IERROR)
28      <type> ORIGIN_ADDR(*)
29      INTEGER(KIND=MPI_ADDRESS_KIND) TARGET_DISP
30      INTEGER ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK, TARGET_COUNT,
31      TARGET_DATATYPE, WIN, REQUEST, IERROR
32
33  MPI_RPUT(ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK,
34           TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, WIN, REQUEST,
35           IERROR)
36      <type> ORIGIN_ADDR(*)
37      INTEGER(KIND=MPI_ADDRESS_KIND) TARGET_DISP
38      INTEGER ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK, TARGET_COUNT,
39      TARGET_DATATYPE, WIN, REQUEST, IERROR
40
41  MPI_WIN_ALLOCATE_SHARED(SIZE, DISP_UNIT, INFO, COMM, BASEPTR, WIN, IERROR)
42      INTEGER DISP_UNIT, INFO, COMM, WIN, IERROR
43      INTEGER(KIND=MPI_ADDRESS_KIND) SIZE, BASEPTR
44
45  If the Fortran compiler provides TYPE(C_PTR), then overloaded by:
46      INTERFACE MPI_WIN_ALLOCATE_SHARED
47          SUBROUTINE MPI_WIN_ALLOCATE_SHARED(SIZE, DISP_UNIT, INFO, COMM, &
48          BASEPTR, WIN, IERROR)
49              IMPORT :: MPI_ADDRESS_KIND
50              INTEGER :: DISP_UNIT, INFO, COMM, WIN, IERROR
51              INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE, BASEPTR

```

#390

```

1  END SUBROUTINE
2  SUBROUTINE MPI_WIN_ALLOCATE_SHARED_CPTR(SIZE, DISP_UNIT, INFO, COMM, &
3      BASEPTR, WIN, IERROR)
4      USE, INTRINSIC :: ISO_C_BINDING, ONLY : C_PTR
5      IMPORT :: MPI_ADDRESS_KIND
6      INTEGER :: DISP_UNIT, INFO, COMM, WIN, IERROR
7      INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE
8      TYPE(C_PTR) :: BASEPTR
9  END SUBROUTINE
10 END INTERFACE
11
12 MPI_WIN_ALLOCATE(SIZE, DISP_UNIT, INFO, COMM, BASEPTR, WIN, IERROR)
13     INTEGER DISP_UNIT, INFO, COMM, WIN, IERROR
14     INTEGER(KIND=MPI_ADDRESS_KIND) SIZE, BASEPTR
15
16 If the Fortran compiler provides TYPE(C_PTR), then overloaded by:
17
18 #390 INTERFACE MPI_WIN_ALLOCATE
19     SUBROUTINE MPI_WIN_ALLOCATE(SIZE, DISP_UNIT, INFO, COMM, BASEPTR, &
20         WIN, IERROR)
21         IMPORT :: MPI_ADDRESS_KIND
22         INTEGER :: DISP_UNIT, INFO, COMM, WIN, IERROR
23         INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE, BASEPTR
24     END SUBROUTINE
25     SUBROUTINE MPI_WIN_ALLOCATE_CPTR(SIZE, DISP_UNIT, INFO, COMM, BASEPTR, &
26         WIN, IERROR)
27         USE, INTRINSIC :: ISO_C_BINDING, ONLY : C_PTR
28         IMPORT :: MPI_ADDRESS_KIND
29         INTEGER :: DISP_UNIT, INFO, COMM, WIN, IERROR
30         INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE
31         TYPE(C_PTR) :: BASEPTR
32     END SUBROUTINE
33 END INTERFACE
34
35 MPI_WIN_ATTACH(WIN, BASE, SIZE, IERROR)
36     INTEGER WIN, IERROR
37     <type> BASE(*)
38     INTEGER (KIND=MPI_ADDRESS_KIND) SIZE
39
40 MPI_WIN_COMPLETE(WIN, IERROR)
41     INTEGER WIN, IERROR
42
43 MPI_WIN_CREATE(BASE, SIZE, DISP_UNIT, INFO, COMM, WIN, IERROR)
44     <type> BASE(*)
45     INTEGER(KIND=MPI_ADDRESS_KIND) SIZE
46     INTEGER DISP_UNIT, INFO, COMM, WIN, IERROR
47
48 MPI_WIN_CREATE_DYNAMIC(INFO, COMM, WIN, IERROR)
49     INTEGER INFO, COMM, WIN, IERROR
50
51 MPI_WIN_DETACH(WIN, BASE, IERROR)
52     INTEGER WIN, IERROR

```

```

1      <type> BASE(*)
2
3      MPI_WIN_FENCE(ASSERT, WIN, IERROR)
4          INTEGER ASSERT, WIN, IERROR
5
6      MPI_WIN_FLUSH_ALL(WIN, IERROR)
7          INTEGER WIN, IERROR
8
9      MPI_WIN_FLUSH_LOCAL_ALL(WIN, IERROR)
10         INTEGER WIN, IERROR
11
12      MPI_WIN_FLUSH_LOCAL(RANK, WIN, IERROR)
13         INTEGER RANK, WIN, IERROR
14
15      MPI_WIN_FLUSH(RANK, WIN, IERROR)
16         INTEGER RANK, WIN, IERROR
17
18      MPI_WIN_FREE(WIN, IERROR)
19         INTEGER WIN, IERROR
20
21      MPI_WIN_GET_GROUP(WIN, GROUP, IERROR)
22         INTEGER WIN, GROUP, IERROR
23
24      MPI_WIN_GET_INFO(WIN, INFO_USED, IERROR)
25         INTEGER WIN, INFO_USED, IERROR
26
27      MPI_WIN_LOCK_ALL(ASSERT, WIN, IERROR)
28         INTEGER ASSERT, WIN, IERROR
29
30      MPI_WIN_LOCK(LOCK_TYPE, RANK, ASSERT, WIN, IERROR)
31         INTEGER LOCK_TYPE, RANK, ASSERT, WIN, IERROR
32
33      MPI_WIN_POST(GROUP, ASSERT, WIN, IERROR)
34         INTEGER GROUP, ASSERT, WIN, IERROR
35
36      MPI_WIN_SET_INFO(WIN, INFO, IERROR)
37         INTEGER WIN, INFO, IERROR
38
39      MPI_WIN_SHARED_QUERY(WIN, RANK, SIZE, DISP_UNIT, BASEPTR, IERROR)
40         INTEGER WIN, RANK, DISP_UNIT, IERROR
41         INTEGER (KIND=MPI_ADDRESS_KIND) SIZE, BASEPTR
42
43      If the Fortran compiler provides TYPE(C_PTR), then overloaded by:
44
45      INTERFACE MPI_WIN_SHARED_QUERY
46      SUBROUTINE MPI_WIN_SHARED_QUERY(WIN, RANK, SIZE, DISP_UNIT, &
47          BASEPTR, IERROR)
48          IMPORT :: MPI_ADDRESS_KIND
49          INTEGER :: WIN, RANK, DISP_UNIT, IERROR
50          INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE, BASEPTR
51      END SUBROUTINE
52
53      SUBROUTINE MPI_WIN_SHARED_QUERY_CPTR(WIN, RANK, SIZE, DISP_UNIT, &
54          BASEPTR, IERROR)
55          USE, INTRINSIC :: ISO_C_BINDING, ONLY : C_PTR
56          IMPORT :: MPI_ADDRESS_KIND

```

#390

```

        INTEGER :: WIN, RANK, DISP_UNIT, IERROR
        INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE
        TYPE(C_PTR) :: BASEPTR
    END SUBROUTINE
END INTERFACE

MPI_WIN_START(GROUP, ASSERT, WIN, IERROR)
    INTEGER GROUP, ASSERT, WIN, IERROR

MPI_WIN_SYNC(WIN, IERROR)
    INTEGER WIN, IERROR

MPI_WIN_TEST(WIN, FLAG, IERROR)
    INTEGER WIN, IERROR
    LOGICAL FLAG

MPI_WIN_UNLOCK_ALL(WIN, IERROR)
    INTEGER WIN, IERROR

MPI_WIN_UNLOCK(RANK, WIN, IERROR)
    INTEGER RANK, WIN, IERROR

MPI_WIN_WAIT(WIN, IERROR)
    INTEGER WIN, IERROR

A.4.10 External Interfaces Fortran Bindings

MPI_GREQUEST_COMPLETE(REQUEST, IERROR)
    INTEGER REQUEST, IERROR

MPI_GREQUEST_START(QUERY_FN, FREE_FN, CANCEL_FN, EXTRA_STATE, REQUEST,
    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)

```