

Distribution Category:  
Mathematics and  
Computer Science (UC-405)

ARGONNE NATIONAL LABORATORY  
9700 South Cass Avenue  
Argonne, IL 60439

---

ANL-00

---

## MPICH Model MPI Implementation Reference Manual

### Draft

by

*William Gropp*  
*Ewing Lusk Mathematics and Computer Science Division*  
*Argonne National Laboratory*

*Nathan Doss*  
*Anthony Skjellum*  
*Department of Computer Science*  
*Mississippi State University*

January 13, 2003

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

### Contents

1	Introduction	1
2	MPI routines	1
3	MPE routines	199

1 Introduction

This document contains detailed documentation on the routines that are part of the MPICH model MPI implementation.  
As an alternate to this manual, the reader should consider using the script `mpiman`; this is a script that uses `xman` to provide a X11 Window System interface to the data in this manual.

2 MPI routines

<code>MPIO_Request_c2f</code>	<code>MPIO_Request_c2f</code>
<code>MPIO_Request_c2f</code> — Translates a C I/O-request handle to a Fortran I/O-request handle	

Synopsis

`MPI_Fint MPIO_Request_c2f(MPIO_Request request)`

Input Parameters

`request` C I/O-request handle (handle)

Return Value

Fortran I/O-request handle (integer)

Location

`./romio/mpi-io/ioreq_c2f.c`

<code>MPIO_Request_f2c</code>	<code>MPIO_Request_f2c</code>
<code>MPIO_Request_f2c</code> — Translates a Fortran I/O-request handle to a C I/O-request handle	

Synopsis

`MPIO_Request MPIO_Request_f2c(MPI_Fint request)`

Input Parameters

`request` Fortran I/O-request handle (integer)

Return Value

C I/O-request handle (handle)

Location

`./romio/mpi-io/ioreq_f2c.c`

<code>MPIO_Test</code>	<code>MPIO_Test</code>
<code>MPIO_Test</code> — Test the completion of a nonblocking read or write	

Synopsis

`int MPIO_Test(MPIO_Request *request, int *flag, MPI_Status *status)`

Input Parameters

`request` request object (handle)

Output Parameters

`flag` true if operation completed (logical)  
`status` status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/iotest.c`

<code>MPIO_Wait</code>	<code>MPIO_Wait</code>
<code>MPIO_Wait</code> — Waits for the completion of a nonblocking read or write	

Synopsis

`int MPIO_Wait(MPIO_Request *request, MPI_Status *status)`

Input Parameters

`request` request object (handle)

Output Parameters

status            status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Location

`./romio/mpi-io/iowait.c`

MPI_Abort	MPI_Abort
MPI_Abort — Terminates MPI execution environment	

Synopsis

int MPI\_Abort( MPI\_Comm comm, int errorcode )

Input Parameters

comm            communicator of tasks to abort  
errorcode       error code to return to invoking environment

Notes

Terminates all MPI processes associated with the communicator `comm`; in most systems (all to date), terminates *all* processes.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Location

`./src/env/abort.c`

MPI_Address	MPI_Address
MPI_Address — Gets the address of a location in memory	

Synopsis

int MPI\_Address( void \*location, MPI\_Aint \*address)

Input Parameters

location        location in caller memory (choice)

Output Parameter

address        address of location (integer)

Note

This routine is provided for both the Fortran and C programmers. On many systems, the address returned by this routine will be the same as produced by the  `C &`  operator, but this is not required in C and may not be true of systems with word- rather than byte-oriented instructions or systems with segmented address spaces.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Location

`./src/pt2pt/address.c`

MPI_Allgather	MPI_Allgather
MPI_Allgather — Gathers data from all tasks and distribute it to all	

Synopsis

int MPI\_Allgather ( void \*sendbuf, int sendcount, MPI\_Datatype sendtype,  
                    void \*recvbuf, int rcvcount, MPI\_Datatype recvtype,  
                    MPI\_Comm comm )

Input Parameters

sendbuf        starting address of send buffer (choice)  
sendcount      number of elements in send buffer (integer)  
sendtype       data type of send buffer elements (handle)  
rcvcount       number of elements received from any process (integer)  
recvtype       data type of receive buffer elements (handle)  
comm           communicator (handle)

## Output Parameter

**recvbuf**      address of receive buffer (choice)

## Notes

The MPI standard (1.0 and 1.1) says that  
 The jth block of data sent from each process is received by every process and placed in the jth block of the buffer **recvbuf**.  
 This is misleading; a better description is  
 The block of data sent from the jth process is received by every process and placed in the jth block of the buffer **recvbuf**.  
 This text was suggested by Rajeev Thakur.

## Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

## Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

### MPL\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

### MPL\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

### MPL\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

### MPL\_ERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

## Location

`./src/coll/allgather.c`

---

### MPLAllgather

---

**MPI\_Allgather**

**MPLAllgather** — Gathers data from all tasks and deliver it to all

## Synopsis

```
int MPI_Allgather ( void *sendbuf, int sendcount, MPI_Datatype sendtype,
                   void *recvbuf, int *recvcounts, int *displs,
                   MPI_Datatype recvtpe, MPI_Comm comm )
```

## Input Parameters

**sendbuf**      starting address of send buffer (choice)  
**sendcount**    number of elements in send buffer (integer)  
**sendtype**      data type of send buffer elements (handle)  
**recvcounts**    integer array (of length group size) containing the number of elements that are received from each process  
**displs**        integer array (of length group size). Entry **i** specifies the displacement (relative to **recvbuf**) at which to place the incoming data from process **i**  
**recvtpe**       data type of receive buffer elements (handle)  
**comm**          communicator (handle)

## Output Parameter

**recvbuf**      address of receive buffer (choice)

## Notes

The MPI standard (1.0 and 1.1) says that

The jth block of data sent from each process is received by every process and placed in the jth block of the buffer **recvbuf**.

This is misleading; a better description is

The block of data sent from the jth process is received by every process and placed in the jth block of the buffer **recvbuf**.

This text was suggested by Rajeev Thakur.

## Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

## Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

### MPL\_ERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

### MPL\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPL\_ERR\_TYPE**  
Invalid datatype argument. May be an uncommitted `MPL_Datatype` (see `MPL_Type_commit`).

**Location**

`./src/coll/allgather.c`

MPL_Allreduce	MPL_Allreduce
---------------	---------------

**MPL\_Allreduce** — Combines values from all processes and distribute the result back to all processes

**Synopsis**

```
int MPI_Allreduce ( void *sendbuf, void *recvbuf, int count,
                   MPI_Datatype datatype, MPI_Op op, MPI_Comm comm )
```

**Input Parameters**

**sendbuf** starting address of send buffer (choice)  
**count** number of elements in send buffer (integer)  
**datatype** data type of elements of send buffer (handle)  
**op** operation (handle)  
**comm** communicator (handle)

**Output Parameter**

**recvbuf** starting address of receive buffer (choice)

**Notes for Fortran**

All MPI routines in Fortran (except for `MPL_WTIME` and `MPL_WTICK`) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g. `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

**Notes on collective operations**

The reduction functions (`MPI_Op`) do not return an error value. As a result, if the functions detect an error, all they can do is either call `MPI_Abort` or silently skip the problem. Thus, if you change the error handler from `MPI_ERRORS_ARE_FATAL` to something else, for example, `MPI_ERRORS_RETURN`, then no error may be indicated.  
The reason for this is the performance problems in ensuring that all collective routines return the same error value.

**Errors**

All MPI routines (except `MPL_Wtime` and `MPL_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPL_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPL\_ERR\_BUFFER**

Invalid buffer pointer. Usually a null buffer where one is not valid.

**MPL\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPL\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPL_Datatype` (see `MPL_Type_commit`).

**MPL\_ERR\_OP**

Invalid operation. MPI operations (objects of type `MPI_Op`) must either be one of the predefined operations (e.g., `MPI_SUM`) or created with `MPI_Op_create`.

**MPL\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**Location**

`./src/coll/allreduce.c`

MPL_Alltoall	MPI_Alltoall
--------------	--------------

**MPI\_Alltoall** — Sends data from all to all processes

**Synopsis**

```
int MPI_Alltoall( void *sendbuf, int sendcount, MPI_Datatype sendtype,
                  void *recvbuf, int rcvcount, MPI_Datatype rcvtype,
                  MPI_Comm comm )
```

**Input Parameters**

**sendbuf** starting address of send buffer (choice)  
**sendcount** number of elements to send to each process (integer)  
**sendtype** data type of send buffer elements (handle)  
**rcvcount** number of elements received from any process (integer)  
**rcvtype** data type of receive buffer elements (handle)  
**comm** communicator (handle)

**Output Parameter**

**recvbuf** address of receive buffer (choice)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPLERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPLERR\_TYPE

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

MPLERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

Location

`./src/coll/alltoall.c`

MPLAlltoallv

**MPLAlltoallv** — Sends data from all to all processes, with a displacement

Synopsis

```
int MPI_Alltoallv (
    void *sendbuf,
    int *sendsizes,
    int *sdispls,
    MPI_Datatype sendtype,
    void *recvbuf,
    int *recvsizes,
    int *rdispls,
    MPI_Datatype recvtype,
    MPI_Comm comm )
```

Input Parameters

`sendbuf` starting address of send buffer (choice)

sendcounts

integer array equal to the group size specifying the number of elements to send to each processor

sdispls

integer array (of length group size). Entry `j` specifies the displacement (relative to `sendbuf` from which to take the outgoing data destined for process `j`)

sendtype

data type of send buffer elements (handle)

recvcounts

integer array equal to the group size specifying the maximum number of elements that can be received from each processor

rdispls

integer array (of length group size). Entry `i` specifies the displacement (relative to `recvbuf` at which to place the incoming data from process `i`)

recvtype

data type of receive buffer elements (handle)

comm

communicator (handle)

Output Parameter

`recvbuf` address of receive buffer (choice)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPLERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPLERR\_TYPE

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

MPLERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

Location

`./src/coll/alltoallv.c`

MPI\_Attr\_delete

**MPI\_Attr\_delete**

**MPI\_Attr\_delete** — Deletes attribute value associated with a key

Synopsis

int MPI\_Attr\_delete ( MPI\_Comm comm, int keyval )

Input Parameters

comm            communicator to which attribute is attached (handle)  
keyval          The key value of the deleted attribute (integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type  **INTEGER**  in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPLERR\_ARG

This error class is associated with an error code that indicates that an attempt was made to free one of the permanent keys.

Location

./src/context/attr\_delval.c

MPI_Attr_get	MPI_Attr_get
MPI_Attr_get — Retrieves attribute value by key	

Synopsis

int MPI\_Attr\_get ( MPI\_Comm comm,  
                  int keyval,  
                  void \*attr\_value,  
                  int \*flag )

Input Parameters

comm            communicator to which attribute is attached (handle)  
keyval          key value (integer)

Output Parameters

attr\_value      attribute value, unless **flag** = false  
flag            true if an attribute value was extracted; false if no attribute is associated with the key

Notes

Attributes must be extracted from the same language as they were inserted in with **MPI\_ATTR\_PUT**. The notes for C and Fortran below explain why.

Notes for C

Even though the **attr\_value** argument is declared as **void \***, it is really the address of a void pointer. See the rationale in the standard for more details.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type  **INTEGER**  in Fortran.

The **attr\_value** in Fortran is a pointer to a Fortran integer, not a pointer to a **void \***.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLSUCCESS

No error; MPI routine completed successfully.

MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPLERR\_OTHER

Other error; the error code associated with this error indicates an attempt to use an invalid keyval.

Location

./src/context/attr\_getval.c

MPI_Atr_put	MPI_Atr_put
MPI_Atr_put — Stores attribute value associated with a key	

Synopsis

int MPI\_Atr\_put ( MPI\_Comm comm, int keyval, void \*attr\_value )

Input Parameters

comm        communicator to which attribute will be attached (handle)  
keyval      key value, as returned by MPI\_KEYVAL\_CREATE (integer)  
attribute\_val   attribute value

Notes

Values of the permanent attributes **MPI\_TAG\_UB**, **MPI\_HOST**, **MPI\_IO**, and **MPI\_WTIME\_IS\_GLOBAL** may not be changed.  
The type of the attribute value depends on whether C or Fortran is being used. In C, an attribute value is a pointer (**void \***); in Fortran, it is a single integer (*not* a pointer, since Fortran has no pointers and there are systems for which a pointer does not fit in an integer (e.g., any > 32 bit address system that uses 64 bits for Fortran **DOUBLE PRECISION**).  
If an attribute is already present, the delete function (specified when the corresponding keyval was created) will be called.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.  
**MPI\_SUCCESS**        No error; MPI routine completed successfully.  
**MPI\_ERR\_COMM**       Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).  
**MPI\_ERR\_OTHER**      Other error; the error code associated with this error indicates an attempt to use an inval keyval.  
**MPI\_ERR\_ARG**

This error class is associated with an error code that indicates that an attempt was made to free one of the permanent keys.

See Also

MPI\_Atr\_get, MPI\_Keyval\_create, MPI\_Atr\_delete

Location

./src/context/attr\_putval.c

MPIBarrier	MPIBarrier
MPIBarrier — Blocks until all process have reached this routine.	

Synopsis

int MPIBarrier ( MPI\_Comm comm )

Input Parameters

comm        communicator (handle)

Notes

Blocks the caller until all group members have called it; the call returns at any process only after all group members have entered the call.

Algorithm

If the underlying device cannot do better, a tree-like or combine algorithm is used to broadcast a message wto all members of the communicator. We can modify this to use "blocks" at a later time (see **MPI\_Bcast**).

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.



## Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

### MPI\_SUCCESS

No error; MPI routine completed successfully.

### MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

## Location

`./src/coll/barrier.c`

### MPI\_Bcast

---

**MPI\_Bcast** — Broadcasts a message from the process with rank "root" to all other processes of the group.

## Synopsis

```
int MPI_Bcast ( void *buffer, int count, MPI_Datatype datatype, int root,
                MPI_Comm comm )
```

## Input/output Parameters

<b>buffer</b>	starting address of buffer (choice)
<b>count</b>	number of entries in buffer (integer)
<b>datatype</b>	data type of buffer (handle)
<b>root</b>	rank of broadcast root (integer)
<b>comm</b>	communicator (handle)

## Algorithm

If the underlying device does not take responsibility, this function uses a tree-like algorithm to broadcast the message to blocks of processes. A linear algorithm is then used to broadcast the message from the first process in a block to all other processes. **MPIR\_BCAST\_BLOCK\_SIZE** determines the size of blocks. If this is set to 1, then this function is equivalent to using a pure tree algorithm. If it is set to the size of the group or greater, it is a pure linear algorithm. The value should be adjusted to determine the most efficient value on different machines.

## Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

## Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

### MPI\_SUCCESS

No error; MPI routine completed successfully.

### MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

### MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

### MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

### MPI\_ERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

### MPI\_ERR\_ROOT

Invalid root. The root must be specified as a rank in the communicator. Ranks must be between zero and the size of the communicator minus one.

## Location

`./src/coll/bcast.c`

### MPI\_Bsend

---

**MPI\_Bsend** — Basic send with user-specified buffering

## Synopsis

```
int MPI_Bsend(
    void *buf,
    int count,
    MPI_Datatype datatype,
    int dest,
    int tag,
    MPI_Comm comm )
```

## Input Parameters

<b>buf</b>	initial address of send buffer (choice)
<b>count</b>	number of elements in send buffer (nonnegative integer)
<b>datatype</b>	datatype of each send buffer element (handle)
<b>dest</b>	rank of destination (integer)

**tag** message tag (integer)  
**comm** communicator (handle)

## Notes

This send is provided as a convenience function; it allows the user to send messages without worrying about where they are buffered (because the user *must* have provided buffer space with `MPI_Buffer_attach`).

In deciding how much buffer space to allocate, remember that the buffer space is not available for reuse by subsequent `MPI_Bsend`s unless you are certain that the message has been received (not just that it should have been received). For example, this code does not allocate enough buffer space

```
MPI_Buffer_attach( b, n*sizeof(double) + MPI_BSEND_OVERHEAD );
for (i=0; i<m; i++) {
    MPI_Bsend( buf, n, MPI_DOUBLE, ... );
}
```

because only enough buffer space is provided for a single send, and the loop may start a second `MPI_Bsend` before the first is done making use of the buffer.

In C, you can force the messages to be delivered by

```
MPI_Buffer_detach( &b, &n );
MPI_Buffer_attach( b, n );
```

(The `MPI_Buffer_detach` will not complete until all buffered messages are delivered.)

## Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

## Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value: C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**  
 No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**  
 Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPI\_ERR\_COUNT**  
 Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_TYPE**  
 Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPI\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Ssendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

**MPI\_ERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Ssendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

## See Also

`MPI_Buffer_attach`, `MPI_Bsend`, `MPI_Bsend_init`

## Location

`./src/pt2pt/bsend.c`

---

**MPI\_Bsend\_init**

---

**MPI\_Bsend\_init** — Builds a handle for a buffered send

## Synopsis

```
int MPI_Bsend_init( void *buf, int count, MPI_Datatype datatype, int dest,
                   int tag, MPI_Comm comm, MPI_Request *request )
```

## Input Parameters

**buf** initial address of send buffer (choice)  
**count** number of elements sent (integer)  
**datatype** type of each element (handle)  
**dest** rank of destination (integer)  
**tag** message tag (integer)  
**comm** communicator (handle)

## Output Parameter

**request** communication request (handle)

## Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

## Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

### MPI\_SUCCESS

No error; MPI routine completed successfully.

### MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

### MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

### MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see `MPI_Type_commit`).

### MPI\_ERR\_RANK

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

### MPI\_ERR\_TAG

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

## Location

`./src/pt2pt/bsend_init.c`

<code>MPI_Buffer_attach</code>	<code>MPI_Buffer_attach</code>
--------------------------------	--------------------------------

**MPI\_Buffer\_attach** — Attaches a user-defined buffer for sending

## Synopsis

```
int MPI_Buffer_attach( void *buffer, int size )
```

## Input Parameters

**buffer**      initial buffer address (choice)  
**size**        buffer size, in bytes (integer)

## Notes

The size given should be the sum of the sizes of all outstanding Bsend's that you intend to have, plus a few hundred bytes for each Bsend that you do. For the purposes of calculating size, you should use `MPI_Pack_size`. In other words, in the code

```
MPI_Buffer_attach( buffer, size );
MPI_Bsend( ..., count=20, datatype=type1, ... );
...
MPI_Bsend( ..., count=40, datatype=type2, ... );
```

the value of `size` in the `MPI_Buffer_attach` call should be greater than the value computed by

```
MPI_Pack_size( 20, type1, comm, &s1 );
MPI_Pack_size( 40, type2, comm, &s2 );
size = s1 + s2 + 2 * MPI_BSEND_OVERHEAD;
```

The `MPI_BSEND_OVERHEAD` gives the maximum amount of space that may be used in the buffer for use by the BSEND routines in using the buffer. This value is in `mpi.h` (for C) and `mpif.h` (for Fortran).

## Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

## Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

### MPI\_SUCCESS

No error; MPI routine completed successfully.

### MPI\_ERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

### MPI\_ERR\_INTERR

An internal error has been detected. This is fatal. Please send a bug report to `mpi-bugs@mcs.anl.gov`.

## See Also

`MPI_Buffer_detach`, `MPI_Bsend`

## Location

`./src/pt2pt/bufattach.c`

<code>MPI_Buffer_detach</code>	<code>MPI_Buffer_detach</code>
<b>MPI_Buffer_detach</b> — Removes an existing buffer (for use in <code>MPI_Bsend</code> etc)	

## Synopsis

```
int MPI_Buffer_detach(
    void *bufferptr,
    int *size )
```

## Output Parameters

**buffer**            initial buffer address (choice)  
**size**              buffer size, in bytes (integer)

## Notes

The reason that `MPI_Buffer_detach` returns the address and size of the buffer being detached is to allow nested libraries to replace and restore the buffer. For example, consider

```
int size, mysize, idummy;
void *ptr, *myptr, *dummy;
MPI_Buffer_detach( &ptr, &size );
MPI_Buffer_attach( myptr, mysize );
...
... library code ...
...
MPI_Buffer_detach( &dummy, &idummy );
MPI_Buffer_attach( ptr, size );
```

This is much like the action of the Unix signal routine and has the same strengths (it is simple) and weaknesses (it only works for nested usages).

Note that for this approach to work, `MPI_Buffer_detach` must return `MPI_SUCCESS` even when there is no buffer to detach. In that case, it returns a size of zero. The MPI 1.1 standard for `MPI_BUFFER_DETACH` contains the text

The statements made in this section describe the behavior of MPI for buffered-mode sends. When no buffer is currently associated, MPI behaves as if a zero-sized buffer is associated with the process.

This could be read as applying only to the various `Bsend` routines. This implementation takes the position that this applies to `MPI_BUFFER_DETACH` as well.

## Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

The Fortran binding for this routine is different. Because Fortran does not have pointers, it is impossible to provide a way to use the output of this routine to exchange buffers. In this case, only the size field is set.

## Notes for C

Even though the `bufferptr` argument is declared as `void *`, it is really the address of a void pointer. See the rationale in the standard for more details.

## Location

`./src/pt2pt/buffree.c`

## MPI\_Cancel

**MPI\_Cancel** — Cancels a communication request

## Synopsis

```
int MPI_Cancel( MPI_Request *request )
```

## Input Parameter

**request**            communication request (handle)

## Note

Cancel has only been implemented for receive requests; it is a no-op for send requests. The primary expected use of `MPI_Cancel` is in multi-buffering schemes, where speculative `MPI_recv`s are made. When the computation completes, some of these receive requests may remain; using `MPI_Cancel` allows the user to cancel these unsatisfied requests.

Cancelling a send operation is much more difficult, in large part because the send will usually be at least partially complete (the information on the tag, size, and source are usually sent immediately to the destination). As of version 1.2.0, MPICH supports cancelling of sends. Users are advised that cancelling a send, while a local operation (as defined by the MPI standard), is likely to be expensive (usually generating one or more internal messages).

## Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

## Null Handles

The MPI 1.1 specification, in the section on opaque objects, explicitly

**disallows freeing a null communicator. The text from the standard is**

A null handle argument is an erroneous IN argument in MPI calls, unless an exception is explicitly stated in the text that defines the function. Such exception is allowed for handles to request objects in `Wait` and `Test` calls (sections `Communication Completion` and `Multiple Completions`). Otherwise, a null handle can only be passed to a function that allocates a new object and returns a reference to it in the handle.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_REQUEST

Invalid **MPI\_Request**. Either null or, in the case of a **MPI\_Start** or **MPI\_Startall**, not a persistent request.

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

Location

`./src/pt2pt/cancel.c`

MPI\_Cart\_coords

MPI\_Cart\_coords

**MPI\_Cart\_coords** — Determines process coords in cartesian topology given rank in group

Synopsis

int MPI\_Cart\_coords ( MPI\_Comm comm, int rank, int maxdims, int \*coords )

Input Parameters

comm        communicator with cartesian structure (handle)  
rank        rank of a process within group of comm (integer)  
maxdims     length of vector coords in the calling program (integer)

Output Parameter

coords       integer array (of size ndims) containing the Cartesian coordinates of specified process (integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_TOPOLOGY

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

MPI\_ERR\_RANK

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

MPI\_ERR\_DIMS

Illegal dimension argument. A dimension argument is null or its length is less than or equal to zero.

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

Location

`./src/topol/cart_coords.c`

MPI\_Cart\_create

MPI\_Cart\_create

**MPI\_Cart\_create** — Makes a new communicator to which topology information has been attached

Synopsis

int MPI\_Cart\_create ( MPI\_Comm comm\_old, int ndims, int \*dims, int \*periods,  
                      int reorder, MPI\_Comm \*comm\_cart )

Input Parameters

comm\_old     input communicator (handle)  
ndims        number of dimensions of cartesian grid (integer)  
dims         integer array of size ndims specifying the number of processes in each dimension  
periods       logical array of size ndims specifying whether the grid is periodic (true) or not  
              (false) in each dimension  
reorder       ranking may be reordered (true) or not (false) (logical)

Output Parameter

comm\_cart    communicator with new cartesian topology (handle)

## Algorithm

We ignore `reorder` info currently.

## Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

## Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

### MPI\_SUCCESS

No error; MPI routine completed successfully.

### MPLERR\_TOPOLOGY

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., `MPI_CART` when expecting `MPI_GRAPH`).

### MPLERR\_DIMS

Illegal dimension argument. A dimension argument is null or its length is less than or equal to zero.

### MPLERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

## Location

`./src/topol/cart_create.c`

### MPI\_Cart\_get

**MPI\_Cart\_get**

**MPI\_Cart\_get** — Retrieves Cartesian topology information associated with a communicator

## Synopsis

```
int MPI_Cart_get (
    MPI_Comm comm,
    int maxdims,
    int *dims,
    int *periods,
    int *coords )
```

## Input Parameters

`comm`      communicator with cartesian structure (handle)  
`maxdims`   length of vectors `dims`, `periods`, and `coords` in the calling program (integer)

## Output Parameters

`dims`      number of processes for each cartesian dimension (array of integer)  
`periods`   periodicity (true/false) for each cartesian dimension (array of logical)  
`coords`    coordinates of calling process in cartesian structure (array of integer)

## Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

## Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

### MPI\_SUCCESS

No error; MPI routine completed successfully.

### MPLERR\_TOPOLOGY

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., `MPI_CART` when expecting `MPI_GRAPH`).

### MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

### MPLERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

## Location

`./src/topol/cart_get.c`

### MPI\_Cart\_map

**MPI\_Cart\_map**

**MPI\_Cart\_map** — Maps process to Cartesian topology information

## Synopsis

```
int MPI_Cart_map (
    MPI_Comm comm_old,
    int ndims,
    int *dims,
```

```
int *periods,
int *newrank)
```

### Input Parameters

**comm** input communicator (handle)  
**ndims** number of dimensions of Cartesian structure (integer)  
**dims** integer array of size **ndims** specifying the number of processes in each coordinate direction  
**periods** logical array of size **ndims** specifying the periodicity specification in each coordinate direction

### Output Parameter

**newrank** reordered rank of the calling process; **MPI\_UNDEFINED** if calling process does not belong to grid (integer)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS** No error; MPI routine completed successfully.

**MPI\_ERR\_COMM** Invalid communicator. A common error is to use a null communicator in a call

(not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_DIMS** Illegal dimension argument. A dimension argument is null or its length is less than or equal to zero.

**MPI\_ERR\_ARG** Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

### Location

`./src/topol/cart_map.c`

---

**MPI\_Cart\_rank**

**MPI\_Cart\_rank**

**MPI\_Cart\_rank** — Determines process rank in communicator given Cartesian location

### Synopsis

```
int MPI_Cart_rank (
MPI_Comm comm,
int *coords,
int *rank )
```

### Input Parameters

**comm** communicator with cartesian structure (handle)  
**coords** integer array (of size **ndims**) specifying the cartesian coordinates of a process

### Output Parameter

**rank** rank of specified process (integer)

### Notes

Out-of-range coordinates are erroneous for non-periodic dimensions. Versions of MPICH before 1.2.2 returned **MPI\_PROC\_NULL** for the rank in this case.

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS** No error; MPI routine completed successfully.

**MPI\_ERR\_TOPOLOGY** Invalid topology. Either there is no topology associated with this communicator,

or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

**MPI\_ERR\_RANK** Invalid source or destination rank. Ranks must be between zero and the size of

the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

**MPI\_ERR\_ARG** Invalid argument. Some argument is invalid and is not identified by a specific

error class (e.g., **MPI\_ERR\_RANK**).

**Location**

`./src/topol/cart_rank.c`

<b>MPL_Cart_shift</b>	<b>MPL_Cart_shift</b>
-----------------------	-----------------------

**MPL\_Cart\_shift** — Returns the shifted source and destination ranks, given a shift direction and amount

**Synopsis**

```
int MPL_Cart_shift ( MPI_Comm comm, int direction, int displ,
                   int *source, int *dest )
```

**Input Parameters**

**comm**        communicator with cartesian structure (handle)  
**direction**   coordinate dimension of shift (integer)  
**displ**        displacement ( $> 0$ : upwards shift,  $< 0$ : downwards shift) (integer)

**Output Parameters**

**rank\_source**   rank of source process (integer)  
**rank\_dest**     rank of destination process (integer)

**Notes**

The **direction** argument is in the range  $[0, n-1]$  for an  $n$ -dimensional Cartesian mesh.

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

**Errors**

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_TOPOLOGY**

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

**MPL\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPL\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

**Location**

`./src/topol/cart_shift.c`

<b>MPL_Cart_sub</b>	<b>MPL_Cart_sub</b>
---------------------	---------------------

**MPL\_Cart\_sub** — Partitions a communicator into subgroups which form lower-dimensional cartesian subgrids

**Synopsis**

```
int MPL_Cart_sub ( MPI_Comm comm, int *remain_dims, MPI_Comm *comm_new )
```

**Input Parameters**

**comm**        communicator with cartesian structure (handle)  
**remain\_dims**   the  $i$ th entry of **remain\_dims** specifies whether the  $i$ th dimension is kept in the subgrid (true) or is dropped (false) (logical vector)

**Output Parameter**

**newcomm**     communicator containing the subgrid that includes the calling process (handle)

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

**Errors**

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_TOPOLOGY**

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).



**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g. **MPI\_ERR\_RANK**).

**Location**

`./src/topol/cart_sub.c`

**MPI\_Cartdim\_get**

**MPI\_Cartdim\_get**

**MPI\_Cartdim\_get** — Retrieves Cartesian topology information associated with a communicator

**Synopsis**

```
int MPI_Cartdim_get ( MPI_Comm comm, int *ndims )
```

**Input Parameter**

**comm**            communicator with cartesian structure (handle)

**Output Parameter**

**ndims**           number of dimensions of the cartesian structure (integer)

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

**Errors**

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g. **MPI\_ERR\_RANK**).

**Location**

`./src/topol/cartdim_get.c`

**MPI\_Comm\_compare**

**MPI\_Comm\_compare**

**MPI\_Comm\_compare** — Compares two communicators

**Synopsis**

```
int MPI_Comm_compare (
    MPI_Comm comm1,
    MPI_Comm comm2,
    int *result)
```

**Input Parameters**

**comm1**           comm1 (handle)  
**comm2**           comm2 (handle)

**Output Parameter**

**result**           integer which is **MPI\_IDENT** if the contexts and groups are the same, **MPI\_CONGRUENT** if different contexts but identical groups, **MPI\_SIMILAR** if different contexts but similar groups, and **MPI\_UNEQUAL** otherwise

**Using 'MPI\_COMM\_NULL' with 'MPI\_Comm\_compare'**

It is an error to use **MPI\_COMM\_NULL** as one of the arguments to **MPI\_Comm\_compare**. The relevant sections of the MPI standard are

(2.4.1 Opaque Objects) A null handle argument is an erroneous IN argument in MPI calls, unless an exception is explicitly stated in the text that defines the function.

(5.4.1. Communicator Accessors) <no text in **MPI\_COMM\_COMPARE** allowing a null handle>

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

**Errors**

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPL\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/context/commcompare.c`

MPI_Comm_create	MPI_Comm_create
MPI_Comm_create — Creates a new communicator	

Synopsis

`int MPI_Comm_create ( MPI_Comm comm, MPI_Group group, MPI_Comm *comm_out )`

Input Parameters

`comm` communicator (handle)  
`group` group, which is a subset of the group of `comm` (handle)

Output Parameter

`comm_out` new communicator (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPI\_ERR\_GROUP

Null group passed to function.

MPI\_ERR\_INTERR

This error is returned when some part of the MPICH implementation is unable to acquire memory.

See Also

`MPI_Comm_free`

Location

`./src/context/comm_create.c`

MPI_Comm_dup	MPI_Comm_dup
MPI_Comm_dup — Duplicates an existing communicator with all its cached information	

Synopsis

`int MPI_Comm_dup (`  
    `MPI_Comm comm,`  
    `MPI_Comm *comm_out )`

Input Parameter

`comm` communicator (handle)

Output Parameter

`newcomm` A new communicator over the same group as `comm` but with a new context. See notes. (handle)

Notes

This routine is used to create a new communicator that has a new communication context but contains the same group of processes as the input communicator. Since all MPI communication is performed within a communicator (specifies as the group of processes *plus* the context), this routine provides an effective way to create a private communicator for use by a software module or library. In particular, no library routine should use `MPI_COMM_WORLD` as the communicator; instead, a duplicate of a user-specified communicator should always be used. For more information, see Using MPI, 2nd edition.  
Because this routine essentially produces a copy of a communicator, it also copies any attributes that have been defined on the input communicator, using the attribute copy function specified by the `copy_function` argument to `MPI_Keyval_create`. This is particularly useful for (a) attributes that describe some property of the group associated with the communicator, such as its interconnection topology and (b) communicators that are given back to the user; the attributes in this case can track subsequent `MPI_Comm_dup` operations on this communicator.



Output Parameters

**namep** One output, contains the name of the communicator. It must be an array of size at least **MPI\_MAX\_NAME\_STRING**.  
**reslen** Number of characters in name

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

Location

`./src/context/comm_name_get.c`

<b>MPI_Comm_group</b>	<b>MPI_Comm_group</b>
<b>MPI_Comm_group</b> — Accesses the group associated with given communicator	

Synopsis

```
int MPI_Comm_group (
    MPI_Comm comm,
    MPI_Group *group )
```

Input Parameter

**comm** Communicator

Output Parameter

**group** Group in communicator

Using 'MPI\_COMM\_NULL' with 'MPI\_Comm\_group'

It is an error to use **MPI\_COMM\_NULL** as one of the arguments to **MPI\_Comm\_group**. The relevant sections of the MPI standard are  
(2.4.1 Opaque Objects) A null handle argument is an erroneous **IN** argument in MPI calls, unless an exception is explicitly stated in the text that defines the function.  
(5.3.2. Group Constructors) <no text in **MPI\_COMM\_GROUP** allowing a null handle>  
Previous versions of MPICH allow **MPI\_COMM\_NULL** in this function. In the interests of promoting portability of applications, we have changed the behavior of **MPI\_Comm\_group** to detect this violation of the MPI standard.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

Location

`./src/context/comm_group.c`

<b>MPI_Comm_rank</b>	<b>MPI_Comm_rank</b>
<b>MPI_Comm_rank</b> — Determines the rank of the calling process in the communicator	

Synopsis

```
int MPI_Comm_rank ( MPI_Comm comm, int *rank )
```

Input Parameters

**comm** communicator (handle)

Output Parameter

rank rank of the calling process in group of comm (integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

`MPI_SUCCESS`

No error; MPI routine completed successfully.

`MPI_ERR_COMM`

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

Location

`./src/context/comm_rank.c`

`MPI_Comm_remote_group`

`MPI_Comm_remote_group`

`MPI_Comm_remote_group` — Accesses the remote group associated with the given inter-communicator

Synopsis

`int MPI_Comm_remote_group ( MPI_Comm comm, MPI_Group *group )`

Input Parameter

comm Communicator (must be intercommunicator)

Output Parameter

group remote group of communicator

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

`MPI_SUCCESS`

No error; MPI routine completed successfully.

`MPI_ERR_COMM`

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

Location

`./src/context/comm_rgroup.c`

`MPI_Comm_remote_size`

`MPI_Comm_remote_size`

`MPI_Comm_remote_size` — Determines the size of the remote group associated with an inter-communicator

Synopsis

`int MPI_Comm_remote_size ( MPI_Comm comm, int *size )`

Input Parameter

comm communicator (handle)

Output Parameter

size number of processes in the group of comm (integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLSUCCESS

No error; MPI routine completed successfully.

MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPLERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/context/comm_rsize.c`

MPI_Comm_set_name	MPI_Comm_set_name
MPI_Comm_set_name — give a print name to the communicator	

Synopsis

int MPI\_Comm\_set\_name( MPI\_Comm com, char \*name )

Input Parameters

com            Communicator to name (handle)  
name          Name for communicator

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLSUCCESS

No error; MPI routine completed successfully.

MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

Location

`./src/context/comm_name_put.c`

MPI_Comm_size	MPI_Comm_size
MPI_Comm_size — Determines the size of the group associated with a communicator	

Synopsis

int MPI\_Comm\_size ( MPI\_Comm comm, int \*size )

Input Parameter

comm            communicator (handle)

Output Parameter

size            number of processes in the group of comm (integer)

Notes

`MPI_COMM_NULL` is *not* considered a valid argument to this function.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLSUCCESS

No error; MPI routine completed successfully.

MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g. `MPI_ERR_RANK`).

**Location**

`./src/context/comm_size.c`

**MPI\_Comm\_split**

**MPI\_Comm\_split**

**MPI\_Comm\_split** — Creates new communicators based on colors and keys

**Synopsis**

`int MPI_Comm_split ( MPI_Comm comm, int color, int key, MPI_Comm *comm_out )`

**Input Parameters**

**comm**                    communicator (handle)  
**color**                   control of subset assignment (nonnegative integer). Processes with the same color are in the same new communicator  
**key**                      control of rank assignment (integer)

**Output Parameter**

**newcomm**                new communicator (handle)

**Notes**

The color must be non-negative or `MPI_UNDEFINED`.

**Notes for Fortran**

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

**Algorithm**

The current algorithm used has quite a few (read: a lot of) inefficiencies that can be removed. Here is what we do for now

- 1) A table is built of colors, and keys (has a next field also).
- 2) The tables of all processes are merged using `{\tt MPI_Allreduce}`.
- 3) Two contexts are allocated for all the comms to be created. These same two contexts can be used for all created communicators since the communicators will not overlap.
- 4) If the local process has a color of `{\tt MPI_UNDEFINED}`, it can return

a `{\tt NULL}` comm.

- 5) The table entries that match the local process color are sorted by `key/rank`.
- 6) A group is created from the sorted list and a communicator is created with this group and the previously allocated contexts.

**Errors**

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPI\_ERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**See Also**

`MPI_Comm_free`

**Location**

`./src/context/comm_split.c`

<code>MPI_Comm_test_inter</code>	<code>MPI_Comm_test_inter</code>
----------------------------------	----------------------------------

**MPI\_Comm\_test\_inter** — Tests to see if a comm is an inter-communicator

**Synopsis**

`int MPI_Comm_test_inter ( MPI_Comm comm, int *flag )`

**Input Parameter**

**comm**                    communicator (handle)

**Output Parameter**

**flag**                    (logical)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_Wtime** and **MPI\_Wtick**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type  **INTEGER**  in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

Location

`./src/context/comm_testvic.c`

MPI_DUP_FN	MPI_DUP_FN
MPI_DUP_FN — A function to simple-mindedly copy attributes	
Location	
<code>./src/context/dup_fn.c</code>	
MPI_Dims_create	MPI_Dims_create
MPI_Dims_create — Creates a division of processors in a cartesian grid	

Synopsis

```
int MPI_Dims_create(  
    int nnodes,  
    int ndims,  
    int *dims)
```

Input Parameters

**nnodes**      number of nodes in a grid (integer)  
**ndims**      number of cartesian dimensions (integer)

In/Out Parameter

**dims**      integer array of size **ndims** specifying the number of nodes in each dimension

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_Wtime** and **MPI\_Wtick**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type  **INTEGER**  in Fortran.

Location

`./src/topol/dims_create.c`

MPI_Errhandler_create	MPI_Errhandler_create
MPI_Errhandler_create — Creates an MPI-style errorhandler	

Synopsis

```
int MPI_Errhandler_create(  
    MPI_Handler_function *function,  
    MPI_Errhandler  
        *errhandler)
```

Input Parameter

**function**      user defined error handling procedure

Output Parameter

**errhandler**      MPI error handler (handle)

Notes

The MPI Standard states that an implementation may make the output value (errhandler) simply the address of the function. However, the action of **MPI\_Errhandler\_free** makes this impossible, since it is required to set the value of the argument to **MPI\_ERRHANDLER\_NULL**. In addition, the actual error handler must remain until all communicators that use it are freed.



Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

`MPI_SUCCESS`

No error; MPI routine completed successfully.

`MPI_ERR_INTERRN`

This error is returned when some part of the MPICH implementation is unable to acquire memory.

Location

`./src/env/errcreate.c`

MPI_Errhandler_free	MPI_Errhandler_free
MPIErrhandler_free — Frees an MPI-style errorhandler	

Synopsis

int MPI\_Errhandler\_free( MPI\_Errhandler \*errhandler )

Input Parameter

errhandler MPI error handler (handle). Set to `MPI_ERRHANDLER_NULL` on exit.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current

MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

`MPI_SUCCESS`

No error; MPI routine completed successfully.

`MPI_ERR_ARG`

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/env/errfree.c`

MPI_Errhandler_get	MPI_Errhandler_get
MPIErrhandler_get — Gets the error handler for a communicator	

Synopsis

int MPI\_Errhandler\_get( MPI\_Comm comm, MPI\_Errhandler \*errhandler )

Input Parameter

comm communicator to get the error handler from (handle)

Output Parameter

errhandler MPI error handler currently associated with communicator (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Note on Implementation

The MPI Standard was unclear on whether this routine required the user to call `MPI_Errhandler_free` once for each call made to this routine in order to free the error handler. After some debate, the MPI Forum added an explicit statement that users are required to call `MPI_Errhandler_free` when the return value from this routine is no longer needed. This behavior is similar to the other MPI routines for getting objects; for example, `MPI_Comm_group` requires that the user call `MPI_Group_free` when the group returned by `MPI_Comm_group` is no longer needed.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/env/errget.c`

MPI\_Errhandler\_set

	MPI_Errhandler_set
--	--------------------

`MPI_Errhandler_set` — Sets the error handler for a communicator

Synopsis

`int MPI_Errhandler_set( MPI_Comm comm, MPI_Errhandler errhandler )`

Input Parameters

`comm`            communicator to set the error handler for (handle)  
`errhandler`    new MPI error handler for communicator (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/env/errset.c`

MPI\_Error\_class

	MPI_Error_class
--	-----------------

`MPI_Error_class` — Converts an error code into an error class

Synopsis

`int MPI_Error_class(  
    int errorcode,  
    int *errorclass)`

Input Parameter

`errorcode`    Error code returned by an MPI routine

Output Parameter

`errorclass`   Error class associated with `errorcode`

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./src/env/errclass.c`

MPI\_Error\_string

	MPI_Error_string
--	------------------

`MPI_Error_string` — Return a string for a given error code

Synopsis

int MPI\_Error\_string( int errorcode, char \*string, int \*resultlen )

Input Parameters

errorcode      Error code returned by an MPI routine or an MPI error class

Output Parameter

string          Text that corresponds to the errorcode  
resultlen       Length of string

Notes: Error codes are the values return by MPI routines (in C) or in the **ierr** argument (in Fortran). These can be converted into error classes with the routine **MPI\_Error\_class**.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./src/env/errorstring.c

MPI_File_c2f	MPI_File_c2f
MPI_File_c2f — Translates a C file handle to a Fortran file handle	

Synopsis

MPI\_Fint MPI\_File\_c2f(MPI\_File fh)

Input Parameters

fh              C file handle (handle)

Return Value

Fortran file handle (integer)

Location

./romio/mpi-io/file\_c2f.c

MPI_File_close	MPI_File_close
MPI_File_close — Closes a file	

Synopsis

int MPI\_File\_close(MPI\_File \*fh)

Input Parameters

fh              file handle (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./romio/mpi-io/close.c

MPI_File_delete	MPI_File_delete
MPI_File_delete — Deletes a file	

Synopsis

int MPI\_File\_delete(char \*filename, MPI\_Info info)

Input Parameters

filename        name of file to delete (string)  
info            info object (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./romio/mpi-io/delete.c

MPI_File_f2c	MPI_File_f2c
--------------	--------------

MPI\_File\_f2c — Translates a Fortran file handle to a C file handle

Synopsis

int MPI\_File\_MPI\_File\_f2c(MPI\_Fint fh)

Input Parameters

fh Fortran file handle (integer)

Return Value

C file handle (handle)

Location

./romio/mpi-io/file\_f2c.c

MPI_File_get_amode	MPI_File_get_amode
--------------------	--------------------

MPI\_File\_get\_amode — Returns the file access mode

Synopsis

int MPI\_File\_get\_amode(MPI\_File fh, int \*amode)

Input Parameters

fh file handle (handle)

Output Parameters

amode access mode (integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./romio/mpi-io/get\_amode.c

MPI_File_get_atomics	MPI_File_get_atomics
----------------------	----------------------

MPI\_File\_get\_atomics — Returns the atomicity mode

Synopsis

int MPI\_File\_get\_atomics(MPI\_File fh, int \*flag)

Input Parameters

fh file handle (handle)

Output Parameters

flag true if atomic mode, false if nonatomic mode (logical)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./romio/mpi-io/get\_atom.c

MPI_File_get_byte_offset	MPI_File_get_byte_offset
--------------------------	--------------------------

MPI\_File\_get\_byte\_offset — Returns the absolute byte position in the file corresponding to "offset" ctypes relative to the current view

Synopsis

int MPI\_File\_get\_byte\_offset(MPI\_File fh, MPI\_Offset offset, MPI\_Offset \*disp)

Input Parameters

fh file handle (handle)  
offset offset (nonnegative integer)

Output Parameters

disp      absolute byte position of offset (nonnegative integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/get_bytoff.c`

---

`MPI_File_get_errhandler`

---

`MPI_File_get_errhandler`

`MPI_File_get_errhandler` — Returns the error handler for a file

Synopsis

int `MPI_File_get_errhandler`(`MPI_File fh`, `MPI_Errhandler *errhandler`)

Input Parameters

`fh`      file handle (handle)

Output Parameters

`errhandler`      error handler (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/get_errh.c`

---

`MPI_File_get_group`

---

`MPI_File_get_group`

`MPI_File_get_group` — Returns the group of processes that opened the file

Synopsis

int `MPI_File_get_group`(`MPI_File fh`, `MPI_Group *group`)

Input Parameters

`fh`      file handle (handle)

Output Parameters

`group`      group that opened the file (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/get_group.c`

---

`MPI_File_get_info`

---

`MPI_File_get_info`

`MPI_File_get_info` — Returns the hints for a file that are actually being used by MPI

Synopsis

int `MPI_File_get_info`(`MPI_File fh`, `MPI_Info *info_used`)

Input Parameters

`fh`      file handle (handle)

Output Parameters

`info_used`      info object (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/get_info.c`

MPI\_File\_get\_position

MPI\_File\_get\_position

**MPI\_File\_get\_position** — Returns the current position of the individual file pointer in etype units relative to the current view

Synopsis

`int MPI_File_get_position(MPI_File fh, MPI_Offset *offset)`

Input Parameters

**fh** file handle (handle)

Output Parameters

**offset** offset of individual file pointer (nonnegative integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/get_posn.c`

MPI\_File\_get\_position\_shared

MPI\_File\_get\_position\_shared

**MPI\_File\_get\_position\_shared** — Returns the current position of the shared file pointer in etype units relative to the current view

Synopsis

`int MPI_File_get_position_shared(MPI_File fh, MPI_Offset *offset)`

Input Parameters

**fh** file handle (handle)

Output Parameters

**offset** offset of shared file pointer (nonnegative integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/get_posn_sh.c`

MPI\_File\_get\_size

MPI\_File\_get\_size

**MPI\_File\_get\_size** — Returns the file size

Synopsis

`int MPI_File_get_size(MPI_File fh, MPI_Offset *size)`

Input Parameters

**fh** file handle (handle)

Output Parameters

**size** size of the file in bytes (nonnegative integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/get_size.c`

MPI\_File\_get\_type\_extent

MPI\_File\_get\_type\_extent

**MPI\_File\_get\_type\_extent** — Returns the extent of datatype in the file

Synopsis

int MPI\_File\_get\_type\_extent(MPI\_File fh, MPI\_Datatype datatype, MPI\_Aint \*extent)

Input Parameters

fh file handle (handle)  
datatype datatype (handle)

Output Parameters

extent extent of the datatype (nonnegative integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/get_extent.c`

MPI_File_get_view	MPI_File_get_view
MPI_File_get_view — Returns the file view	

Synopsis

int MPI\_File\_get\_view(MPI\_File fh, MPI\_Offset \*disp, MPI\_Datatype \*etype, MPI\_Datatype \*filetype, char \*datarep)

Input Parameters

fh file handle (handle)

Output Parameters

disp displacement (nonnegative integer)  
etype elementary datatype (handle)  
filetype filetype (handle)  
datarep data representation (string)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/get_view.c`

MPI_File_iread	MPI_File_iread
MPI_File_iread — Nonblocking read using individual file pointer	

Synopsis

int MPI\_File\_iread(MPI\_File fh, void \*buf, int count, MPI\_Datatype datatype, MPI\_Request \*request)

Input Parameters

fh file handle (handle)  
count number of elements in buffer (nonnegative integer)  
datatype datatype of each buffer element (handle)

Output Parameters

buf initial address of buffer (choice)  
request request object (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/iread.c`

MPI_File_iread_at	MPI_File_iread_at
MPI_File_iread_at — Nonblocking read using explicit offset	

Synopsis

```
int MPI_File_iread_at(MPI_File fh, MPI_Offset offset, void *buf,  
                     int count, MPI_Datatype datatype,  
                     MPIO_Request *request)
```

Input Parameters

**fh** file handle (handle)  
**offset** file offset (nonnegative integer)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**buf** initial address of buffer (choice)  
**request** request object (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/iread_at.c`

MPI_File_iread_shared	MPI_File_iread_shared
MPI_File_iread_shared — Nonblocking read using shared file pointer	

Synopsis

```
int MPI_File_iread_shared(MPI_File fh, void *buf, int count,  
                          MPI_Datatype datatype, MPIO_Request *request)
```

Input Parameters

**fh** file handle (handle)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**buf** initial address of buffer (choice)  
**request** request object (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/iread_sh.c`

MPI_File_irewrite	MPI_File_irewrite
MPI_File_irewrite — Nonblocking write using individual file pointer	

Synopsis

```
int MPI_File_irewrite(MPI_File fh, void *buf, int count,  
                      MPI_Datatype datatype, MPIO_Request *request)
```

Input Parameters

**fh** file handle (handle)  
**buf** initial address of buffer (choice)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**request** request object (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/irewrite.c`



MPI_File_iwrite_at	MPI_File_iwrite_at
MPI_File_iwrite_at — Nonblocking write using explicit offset	

Synopsis

```
int MPI_File_iwrite_at(MPI_File fh, MPI_Offset offset, void *buf,  
                      int count, MPI_Datatype datatype,  
                      MPIO_Request *request)
```

Input Parameters

**fh** file handle (handle)  
**offset** file offset (nonnegative integer)  
**buf** initial address of buffer (choice)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**request** request object (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/iwrite_at.c`

MPI_File_iwrite_shared	MPI_File_iwrite_shared
MPI_File_iwrite_shared — Nonblocking write using shared file pointer	

Synopsis

```
int MPI_File_iwrite_shared(MPI_File fh, void *buf, int count,  
                          MPI_Datatype datatype, MPIO_Request *request)
```

Input Parameters

**fh** file handle (handle)  
**buf** initial address of buffer (choice)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**request** request object (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/iwrite_sh.c`

MPI_File_open	MPI_File_open
MPI_File_open — Opens a file	

Synopsis

```
int MPI_File_open(MPI_Comm comm, char *filename, int amode,  
                 MPI_Info info, MPI_File *fh)
```

Input Parameters

**comm** communicator (handle)  
**filename** name of file to open (string)  
**amode** file access mode (integer)  
**info** info object (handle)

Output Parameters

**fh** file handle (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./romio/mpi-io/open.c

MPL\_File\_preallocate

MPL\_File\_preallocate

MPL\_File\_preallocate — Preallocates storage space for a file

Synopsis

int MPI\_File\_preallocate(MPI\_File fh, MPI\_Offset size)

Input Parameters

fh file handle (handle)  
size size to preallocate (nonnegative integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./romio/mpi-io/prealloc.c

MPL\_File\_read

MPL\_File\_read

MPL\_File\_read — Read using individual file pointer

Synopsis

int MPI\_File\_read(MPI\_File fh, void \*buf, int count,  
MPI\_Datatype datatype, MPI\_Status \*status)

Input Parameters

fh file handle (handle)  
count number of elements in buffer (nonnegative integer)  
datatype datatype of each buffer element (handle)

Output Parameters

buf initial address of buffer (choice)  
status status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./romio/mpi-io/read.c

MPL\_File\_read\_all

MPL\_File\_read\_all

MPL\_File\_read\_all — Collective read using individual file pointer

Synopsis

int MPI\_File\_read\_all(MPI\_File fh, void \*buf, int count,  
MPI\_Datatype datatype, MPI\_Status \*status)

Input Parameters

fh file handle (handle)  
count number of elements in buffer (nonnegative integer)  
datatype datatype of each buffer element (handle)

Output Parameters

buf initial address of buffer (choice)  
status status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./romio/mpi-io/read\_all.c

MPI_File_read_all_begin	MPI_File_read_all_end
MPI_File_read_all_begin — Begin a split collective read using individual file pointer	

Synopsis

```
int MPI_File_read_all_begin(MPI_File fh, void *buf, int count,
                             MPI_Datatype datatype)
```

Input Parameters

**fh** file handle (handle)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**buf** initial address of buffer (choice)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/read_allb.c`

MPI_File_read_all_end	MPI_File_read_all_end
MPI_File_read_all_end — Complete a split collective read using individual file pointer	

Synopsis

```
int MPI_File_read_all_end(MPI_File fh, void *buf, MPI_Status *status)
```

Input Parameters

**fh** file handle (handle)

Output Parameters

**buf** initial address of buffer (choice)  
**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/read_allc.c`

MPI_File_read_at	MPI_File_read_at
MPI_File_read_at — Read using explicit offset	

Synopsis

```
int MPI_File_read_at(MPI_File fh, MPI_Offset offset, void *buf,
                     int count, MPI_Datatype datatype, MPI_Status *status)
```

Input Parameters

**fh** file handle (handle)  
**offset** file offset (nonnegative integer)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**buf** initial address of buffer (choice)  
**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/read_at.c`

MPI_File_read_at_all	MPI_File_read_at_all
MPI_File_read_at_all — Collective read using explicit offset	

Synopsis

```
int MPI_File_read_at_all(MPI_File fh, MPI_Offset offset, void *buf,
                        int count, MPI_Datatype datatype,
                        MPI_Status *status)
```

Input Parameters

**fh** file handle (handle)  
**offset** file offset (nonnegative integer)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**buf** initial address of buffer (choice)  
**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/read_atall.c`

MPI_File_read_at_all_begin	MPI_File_read_at_all_begin
MPI_File_read_at_all_begin — Begin a split collective read using explicit offset	

Synopsis

```
int MPI_File_read_at_all_begin(MPI_File fh, MPI_Offset offset, void *buf,
                              int count, MPI_Datatype datatype)
```

Input Parameters

**fh** file handle (handle)  
**offset** file offset (nonnegative integer)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**buf** initial address of buffer (choice)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/rd_atallb.c`

MPI_File_read_at_all_end	MPI_File_read_at_all_end
MPI_File_read_at_all_end — Complete a split collective read using explicit offset	

Synopsis

```
int MPI_File_read_at_all_end(MPI_File fh, void *buf, MPI_Status *status)
```

Input Parameters

**fh** file handle (handle)

Output Parameters

**buf** initial address of buffer (choice)  
**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/rd_atalle.c`

MPI_File_read_ordered	MPI_File_read_ordered
MPI_File_read_ordered — Collective read using shared file pointer	

Synopsis

int MPI\_File\_read\_ordered(MPI\_File fh, void \*buf, int count,  
MPI\_Datatype datatype, MPI\_Status \*status)

Input Parameters

fh file handle (handle)  
count number of elements in buffer (nonnegative integer)  
datatype datatype of each buffer element (handle)

Output Parameters

buf initial address of buffer (choice)  
status status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./romio/mpi-io/read\_ord.c

MPI_File_read_ordered_begin	MPI_File_read_ordered_begin
MPI_File_read_ordered_begin — Begin a split collective read using shared file pointer	

Synopsis

int MPI\_File\_read\_ordered\_begin(MPI\_File fh, void \*buf, int count,  
MPI\_Datatype datatype)

Input Parameters

fh file handle (handle)  
count number of elements in buffer (nonnegative integer)  
datatype datatype of each buffer element (handle)

Output Parameters

buf initial address of buffer (choice)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./romio/mpi-io/read\_orde.c

MPI_File_read_ordered_end	MPI_File_read_ordered_end
MPI_File_read_ordered_end — Complete a split collective read using shared file pointer	

Synopsis

int MPI\_File\_read\_ordered\_end(MPI\_File fh, void \*buf, MPI\_Status \*status)

Input Parameters

fh file handle (handle)

Output Parameters

buf initial address of buffer (choice)  
status status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./romio/mpi-io/read\_orde.c

MPI_File_read_shared	MPI_File_read_shared
MPI_File_read_shared — Read using shared file pointer	

Synopsis

int MPI\_File\_read\_shared(MPI\_File fh, void \*buf, int count, MPI\_Datatype datatype, MPI\_Status \*status)

Input Parameters

- fh file handle (handle)
- count number of elements in buffer (nonnegative integer)
- datatype datatype of each buffer element (handle)

Output Parameters

- buf initial address of buffer (choice)
- status status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Location

./romio/mpi-io/read\_sh.c

MPI\_File\_seek

MPI\_File\_seek

MPI\_File\_seek — Updates the individual file pointer

Synopsis

int MPI\_File\_seek(MPI\_File fh, MPI\_Offset offset, int whence)

Input Parameters

- fh file handle (handle)
- offset file offset (integer)
- whence update mode (state)

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Location

./romio/mpi-io/seek.c

MPI\_File\_seek\_shared

MPI\_File\_seek\_shared

MPI\_File\_seek\_shared — Updates the shared file pointer

Synopsis

int MPI\_File\_seek\_shared(MPI\_File fh, MPI\_Offset offset, int whence)

Input Parameters

- fh file handle (handle)
- offset file offset (integer)
- whence update mode (state)

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Location

./romio/mpi-io/seek\_sh.c

MPI\_File\_set\_atomics

MPI\_File\_set\_atomics

MPI\_File\_set\_atomics — Sets the atomicity mode

Synopsis

int MPI\_File\_set\_atomics(MPI\_File fh, int flag)

Input Parameters

- fh file handle (handle)
- flag true to set atomic mode, false to set nonatomic mode (logical)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/set_atom.c`

MPI_File_set_errhandler	MPI_File_set_errhandler
MPI_File_set_errhandler — Sets the error handler for a file	

Synopsis

`int MPI_File_set_errhandler(MPI_File fh, MPI_Errhandler errhandler)`

Input Parameters

**fh** file handle (handle)  
**errhandler** error handler (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/set_errh.c`

MPI_File_set_info	MPI_File_set_info
MPI_File_set_info — Sets new values for the hints associated with a file	

Synopsis

`int MPI_File_set_info(MPI_File fh, MPI_Info info)`

Input Parameters

**fh** file handle (handle)  
**info** info object (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/set_info.c`

MPI_File_set_size	MPI_File_set_size
MPI_File_set_size — Sets the file size	

Synopsis

`int MPI_File_set_size(MPI_File fh, MPI_Offset size)`

Input Parameters

**fh** file handle (handle)  
**size** size to truncate or expand file (nonnegative integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./romio/mpi-io/set_size.c`

MPI_File_set_view	MPI_File_set_view
MPI_File_set_view — Sets the file view	

Synopsis

int MPI\_File\_set\_view(MPI\_File fh, MPI\_Offset disp, MPI\_Datatype etype, MPI\_Datatype filetype, char \*datarep, MPI\_Info info)

Input Parameters

- fh file handle (handle)
- disp displacement (nonnegative integer)
- etype elementary datatype (handle)
- filetype filetype (handle)
- datarep data representation (string)
- info info object (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type  **INTEGER**  in Fortran.

Location

./romio/mpi-io/set\_view.c

MPI_File_sync	MPI_File_sync
MPI_File_sync — Causes all previous writes to be transferred to the storage device	

Synopsis

int MPI\_File\_sync(MPI\_File fh)

Input Parameters

- fh file handle (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type  **INTEGER**  in Fortran..

Location

./romio/mpi-io/fsync.c

MPI_File_write	MPI_File_write
MPI_File_write — Write using individual file pointer	

Synopsis

int MPI\_File\_write(MPI\_File fh, void \*buf, int count, MPI\_Datatype datatype, MPI\_Status \*status)

Input Parameters

- fh file handle (handle)
- buf initial address of buffer (choice)
- count number of elements in buffer (nonnegative integer)
- datatype datatype of each buffer element (handle)

Output Parameters

- status status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type  **INTEGER**  in Fortran.

Location

./romio/mpi-io/write.c

MPI_File_write_all	MPI_File_write_all
MPI_File_write_all — Collective write using individual file pointer	

Synopsis

int MPI\_File\_write\_all(MPI\_File fh, void \*buf, int count, MPI\_Datatype datatype, MPI\_Status \*status)



Input Parameters

**fh** file handle (handle)  
**buf** initial address of buffer (choice)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_all.c`

---

**MPI\_File\_write\_all\_begin**

---

**MPI\_File\_write\_all\_begin** — Begin a split collective write using individual file pointer

Synopsis

```
int MPI_File_write_all_begin(MPI_File fh, void *buf, int count,  
                             MPI_Datatype datatype)
```

Input Parameters

**fh** file handle (handle)  
**buf** initial address of buffer (choice)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_allb.c`

---

**MPI\_File\_write\_all\_end**

---

**MPI\_File\_write\_all\_end** — Complete a split collective write using individual file pointer

Synopsis

```
int MPI_File_write_all_end(MPI_File fh, void *buf, MPI_Status *status)
```

Input Parameters

**fh** file handle (handle)

Output Parameters

**buf** initial address of buffer (choice)  
**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_allb.c`

---

**MPI\_File\_write\_at**

---

**MPI\_File\_write\_at** — Write using explicit offset

Synopsis

```
int MPI_File_write_at(MPI_File fh, MPI_Offset offset, void *buf,  
                      int count, MPI_Datatype datatype,  
                      MPI_Status *status)
```

Input Parameters

**fh** file handle (handle)  
**offset** file offset (nonnegative integer)  
**buf** initial address of buffer (choice)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_at.c`

MPI_File_write_at_all	MPI_File_write_at_all
MPI_File_write_at_all — Collective write using explicit offset	

Synopsis

```
int MPI_File_write_at_all(MPI_File fh, MPI_Offset offset, void *buf,  
                          int count, MPI_Datatype datatype,  
                          MPI_Status *status)
```

Input Parameters

**fh** file handle (handle)  
**offset** file offset (nonnegative integer)  
**buf** initial address of buffer (choice)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Output Parameters

**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_atall.c`

MPI_File_write_at_all_begin	MPI_File_write_at_all_begin
MPI_File_write_at_all_begin — Begin a split collective write using explicit offset	

Synopsis

```
int MPI_File_write_at_all_begin(MPI_File fh, MPI_Offset offset, void *buf,  
                               int count, MPI_Datatype datatype)
```

Input Parameters

**fh** file handle (handle)  
**offset** file offset (nonnegative integer)  
**buf** initial address of buffer (choice)  
**count** number of elements in buffer (nonnegative integer)  
**datatype** datatype of each buffer element (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/wr_atallb.c`

MPI_File_write_at_all_end	MPI_File_write_at_all_end
MPI_File_write_at_all_end — Complete a split collective write using explicit offset	

Synopsis

```
int MPI_File_write_at_all_end(MPI_File fh, void *buf, MPI_Status *status)
```

Input Parameters

**fh** file handle (handle)  
**buf** initial address of buffer (choice)

Output Parameters

**status** status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/wr_atallie.c`

MPI_File_write_ordered		MPI_File_write_ordered
MPI_File_write_ordered — Collective write using shared file pointer		
Synopsis		
int MPI_File_write_ordered(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)		
Input Parameters		
<b>fh</b>	file handle (handle)	
<b>buf</b>	initial address of buffer (choice)	
<b>count</b>	number of elements in buffer (nonnegative integer)	
<b>datatype</b>	datatype of each buffer element (handle)	
Output Parameters		
<b>status</b>	status object (Status)	

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_ordb.c`

MPI_File_write_ordered_begin		MPI_File_write_ordered_begin
MPI_File_write_ordered_begin — Begin a split collective write using shared file pointer		
Synopsis		
int MPI_File_write_ordered_begin(MPI_File fh, void *buf, int count, MPI_Datatype datatype)		
Input Parameters		
<b>fh</b>	file handle (handle)	
<b>count</b>	number of elements in buffer (nonnegative integer)	
<b>datatype</b>	datatype of each buffer element (handle)	
Output Parameters		
<b>buf</b>	initial address of buffer (choice)	

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_ordb.c`

MPI_File_write_ordered_end		MPI_File_write_ordered_end
MPI_File_write_ordered_end — Complete a split collective write using shared file pointer		
Synopsis		
int MPI_File_write_ordered_end(MPI_File fh, void *buf, MPI_Status *status)		
Input Parameters		
<b>fh</b>	file handle (handle)	

Output Parameters

**buf**            initial address of buffer (choice)  
**status**        status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_orde.c`

MPI_File_write_shared		MPI_File_write_shared
MPI_File_write_shared — Write using shared file pointer		
Synopsis		
int MPI_File_write_shared(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)		
Input Parameters		
<b>fh</b>	file handle (handle)	
<b>buf</b>	initial address of buffer (choice)	
<b>count</b>	number of elements in buffer (nonnegative integer)	
<b>datatype</b>	datatype of each buffer element (handle)	

Output Parameters

**status**        status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./romio/mpi-io/write_sh.c`

MPI_Finalize	MPI_Finalize
MPI_Finalize — Terminates MPI execution environment	

Synopsis

int MPI\_Finalize()

Notes

All processes must call this routine before exiting. The number of processes running *after* this routine is called is undefined; it is best not to perform much more than a **return rc** after calling **MPI\_Finalize**.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./src/env/finalize.c`

MPI_Finalized	MPI_Finalized
MPI_Finalized — Indicates whether MPI_Finalize has been called.	

Synopsis

int MPI\_Finalized( int \*flag )

Output Parameter

**flag**            Flag is true if **MPI\_Finalize** has been called and false otherwise.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

**Location**

./src/misc2/finalized.c

**MPL\_Gather****MPL\_Gather** — Gathers together values from a group of processes**Synopsis**

```
int MPI_Gather ( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int recvcnt, MPI_Datatype recvttype,
                int root, MPI_Comm comm )
```

**Input Parameters**

**sendbuf** starting address of send buffer (choice)  
**sendcount** number of elements in send buffer (integer)  
**sendtype** data type of send buffer elements (handle)  
**recvcnt** number of elements for any single receive (integer, significant only at root)  
**recvttype** data type of recv buffer elements (significant only at root) (handle)  
**root** rank of receiving process (integer)  
**comm** communicator (handle)

**Output Parameter**

**recvbuf** address of receive buffer (choice, significant only at **root**)

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_Wtime** and **MPI\_Wtick**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

**Errors**

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see **MPI\_Type\_commit**).

**MPI\_ERR\_BUFFER**

Invalid buffer pointer. Usually a null buffer where one is not valid.

**Location**

./src/coll/gather.c

**MPL\_Gatherv****MPL\_Gatherv** — Gathers into specified locations from all processes in a group**Synopsis**

```
int MPI_Gatherv ( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                  void *recvbuf, int *recvcounts, int *displs,
                  MPI_Datatype recvttype,
                  int root, MPI_Comm comm )
```

**Input Parameters**

**sendbuf** starting address of send buffer (choice)  
**sendcount** number of elements in send buffer (integer)  
**sendtype** data type of send buffer elements (handle)  
**recvcounts** integer array (of length group size) containing the number of elements that are received from each process (significant only at **root**)  
**displs** integer array (of length group size). Entry **i** specifies the displacement relative to **recvbuf** at which to place the incoming data from process **i** (significant only at **root**)  
**recvttype** data type of recv buffer elements (significant only at **root**) (handle)  
**root** rank of receiving process (integer)  
**comm** communicator (handle)

**Output Parameter**

**recvbuf** address of receive buffer (choice, significant only at **root**)

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_Wtime** and **MPI\_Wtick**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLSUCCESS

No error; MPI routine completed successfully.

MPLERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPLERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

MPLERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

Location

./src/coll/gatherv.c

MPI_Get_count	MPI_Get_count
MPI_Get_count — Gets the number of "top level" elements	

Synopsis

```
int MPI_Get_count(  
    MPI_Status *status,  
    MPI_Datatype datatype,  
    int *count )
```

Input Parameters

status      return status of receive operation (Status)  
datatype    datatype of each receive buffer element (handle)

Output Parameter

count      number of received elements (integer) Notes: If the size of the datatype is zero, this routine will return a count of zero. If the amount of data in **status** is not an exact multiple of the size of **datatype** (so that **count** would not be integral), a count of **MPI\_UNDEFINED** is returned instead.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return

value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPLSUCCESS

No error; MPI routine completed successfully.

MPLERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

Location

./src/pt2pt/getcount.c

MPI_Get_elements	MPI_Get_elements
MPI_Get_elements — Returns the number of basic elements in a datatype	

Synopsis

```
int MPI_Get_elements ( MPI_Status *status, MPI_Datatype datatype,  
    int *elements )
```

Input Parameters

status      return status of receive operation (Status)  
datatype    datatype used by receive operation (handle)

Output Parameter

count      number of received basic elements (integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.



### Input Parameters

**comm\_old** input communicator without topology (handle)  
**nnodes** number of nodes in graph (integer)  
**index** array of integers describing node degrees (see below)  
**edges** array of integers describing graph edges (see below)  
**reorder** ranking may be reordered (true) or not (false) (logical)

### Output Parameter

**comm\_graph** communicator with graph topology added (handle)

### Algorithm

We ignore the **reorder** info currently.

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_TOPOLOGY**

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

### Location

`./src/topol/graphcreate.c`

---

**MPI\_Graph\_get**

**MPI\_Graph\_get**

---

**MPI\_Graph\_get** — Retrieves graph topology information associated with a communicator

### Synopsis

```
int MPI_Graph_get ( MPI_Comm comm, int maxindex, int maxedges,
                  int *index, int *edges )
```

### Input Parameters

**comm** communicator with graph structure (handle)  
**maxindex** length of vector **index** in the calling program (integer)  
**maxedges** length of vector **edges** in the calling program (integer)

### Output Parameter

**index** array of integers containing the graph structure (for details see the definition of **MPI\_GRAPH\_CREATE**)  
**edges** array of integers containing the graph structure

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_TOPOLOGY**

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

### Location

`./src/topol/graph_get.c`

---

**MPI\_Graph\_map**

**MPI\_Graph\_map**

---

**MPI\_Graph\_map** — Maps process to graph topology information



## Synopsis

```
int MPI_Graph_map ( MPI_Comm comm_old, int nnodes, int *index, int *edges,
                  int *newrank )
```

## Input Parameters

**comm** input communicator (handle)  
**nnodes** number of graph nodes (integer)  
**index** integer array specifying the graph structure; see **MPI\_GRAPH\_CREATE**  
**edges** integer array specifying the graph structure

## Output Parameter

**newrank** reordered rank of the calling process; **MPI\_UNDEFINED** if the calling process does not belong to graph (integer)

## Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

## Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_TOPOLOGY**

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

**MPLERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPLERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

## Location

`./src/topol/graph_map.c`

---

**MPI\_Graph\_neighbors**

**MPI\_Graph\_neighbors**

**MPI\_Graph\_neighbors** — Returns the neighbors of a node associated with a graph topology

## Synopsis

```
int MPI_Graph_neighbors ( MPI_Comm comm, int rank, int maxneighbors,
                        int *neighbors )
```

## Input Parameters

**comm** communicator with graph topology (handle)  
**rank** rank of process in group of comm (integer)  
**maxneighbors** size of array neighbors (integer)

## Output Parameters

**neighbors** ranks of processes that are neighbors to specified process (array of integer)

## Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

## Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPLSUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_TOPOLOGY**

Invalid topology. Either there is no topology associated with this communicator, or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

**MPLERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPLERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

**MPLERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

## Location

`./src/topol/graph_nbr.c`

---

**MPI\_Graph\_neighbors\_count**


---

**MPI\_Graph\_neighbors\_count** — Returns the number of neighbors of a node associated with a graph topology

### Synopsis

```
int MPI_Graph_neighbors_count ( MPI_Comm comm, int rank, int *nneighbors )
```

### Input Parameters

**comm**      communicator with graph topology (handle)  
**rank**      rank of process in group of **comm** (integer)

### Output Parameter

**nneighbors**    number of neighbors of specified process (integer)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_Wtime** and **MPI\_Wtick**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**      No error; MPI routine completed successfully.

**MPL\_ERR\_TOPOLOGY**    Invalid topology. Either there is no topology associated with this communicator,

or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

**MPL\_ERR\_COMM**      Invalid communicator. A common error is to use a null communicator in a call

(not even allowed in **MPI\_Comm\_rank**).

**MPL\_ERR\_ARG**      Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

**MPL\_ERR\_RANK**      Invalid source or destination rank. Ranks must be between zero and the size of

the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

### Location

`./src/topol/graphnbrcnt.c`

---

**MPI\_Graphdims\_get**


---



---

**MPI\_Graphdims\_get**


---

**MPI\_Graphdims\_get** — Retrieves graph topology information associated with a communicator

### Synopsis

```
int MPI_Graphdims_get ( MPI_Comm comm, int *nnodes, int *nedges )
```

### Input Parameters

**comm**      communicator for group with graph structure (handle)

### Output Parameter

**nnodes**      number of nodes in graph (integer)  
**nedges**      number of edges in graph (integer)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_Wtime** and **MPI\_Wtick**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**      No error; MPI routine completed successfully.

**MPL\_ERR\_TOPOLOGY**    Invalid topology. Either there is no topology associated with this communicator,

or it is not the correct type (e.g., **MPI\_CART** when expecting **MPI\_GRAPH**).

**MPL\_ERR\_COMM**      Invalid communicator. A common error is to use a null communicator in a call

(not even allowed in **MPI\_Comm\_rank**).

**MPL\_ERR\_ARG**      Invalid argument. Some argument is invalid and is not identified by a specific

error class (e.g., **MPI\_ERR\_RANK**).

Location

./src/topo1/graphdimsget.c

MPI_Group_compare	MPI_Group_compare
-------------------	-------------------

MPI\_Group\_compare — Compares two groups

Synopsis

int MPI\_Group\_compare ( MPI\_Group group1, MPI\_Group group2, int \*result )

Input Parameters

group1 (handle)  
group2 (handle)

Output Parameter

result integer which is MPI\_IDENT if the order and members of the two groups are the same, MPI\_SIMILAR if only the members are the same, and MPI\_UNEQUAL otherwise

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Errors

All MPI routines (except MPI\_Wtime and MPI\_Wtick) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with MPI\_Errhandler\_set; the predefined error handler MPI\_ERRORS\_RETURN may be used to cause error values to be returned. Note that MPI does not guarantee that an MPI program can continue past an error.  
MPI\_SUCCESS No error; MPI routine completed successfully.  
MPI\_ERR\_GROUP Null group passed to function.  
MPI\_ERR\_ARG Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., MPI\_ERR\_RANK).

Location

./src/context/groupcompare.c

MPI_Group_difference	MPI_Group_difference
----------------------	----------------------

MPI\_Group\_difference — Makes a group from the difference of two groups

Synopsis

int MPI\_Group\_difference ( MPI\_Group group1, MPI\_Group group2, MPI\_Group \*group\_out )

Input Parameters

group1 first group (handle)  
group2 second group (handle)

Output Parameter

newgroup difference group (handle)

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Errors

All MPI routines (except MPI\_Wtime and MPI\_Wtick) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with MPI\_Errhandler\_set; the predefined error handler MPI\_ERRORS\_RETURN may be used to cause error values to be returned. Note that MPI does not guarantee that an MPI program can continue past an error.  
MPI\_SUCCESS No error; MPI routine completed successfully.  
MPI\_ERR\_GROUP Null group passed to function.  
MPI\_ERR\_INTEN This error is returned when some part of the MPICH implementation is unable to acquire memory.

See Also

MPI\_Group\_free

Location

./src/context/group\_diff.c

MPI_Group_excl	MPI_Group_excl
----------------	----------------

**MPI\_Group\_excl** — Produces a group by reordering an existing group and taking only unlisted members

Synopsis

```
int MPI_Group_excl ( MPI_Group group, int n, int *ranks, MPI_Group *newgroup )
```

Input Parameters

**group** group (handle)  
**n** number of elements in array **ranks** (integer)  
**ranks** array of integer ranks in **group** not to appear in **newgroup**

Output Parameter

**newgroup** new group derived from above, preserving the order defined by **group** (handle)

Note

Currently, each of the ranks to exclude must be a valid rank in the group and all elements must be distinct or the function is erroneous. This restriction is per the draft.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_GROUP**

Null group passed to function.

**MPI\_ERR\_INTEN**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

**MPI\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

See Also

**MPI\_Group\_free**

Location

./src/context/group\_excl.c

MPI_Group_free	MPI_Group_free
----------------	----------------

**MPI\_Group\_free** — Frees a group

Synopsis

```
int MPI_Group_free ( MPI_Group *group )
```

Input Parameter  
**group** group (handle)

Notes

On output, **group** is set to **MPI\_GROUP\_NULL**.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPL\_ERR\_RANK**).

**MPL\_ERR\_ARG**

This error class is associated with an error code that indicates that an attempt was made to free one of the permanent groups.

**Location**

`./src/context/group_free.c`

**MPL\_Group\_incl**

	MPL_Group_incl
--	----------------

**MPL\_Group\_incl** — Produces a group by reordering an existing group and taking only listed members

**Synopsis**

`int MPI_Group_incl ( MPI_Group group, int n, int *ranks, MPI_Group *group_out )`

**Input Parameters**

**group** group (handle)  
**n** number of elements in array **ranks** (and size of **newgroup**) (integer)  
**ranks** ranks of processes in **group** to appear in **newgroup** (array of integers)

**Output Parameter**

**newgroup** new group derived from above, in the order defined by **ranks** (handle)

**Note**

This implementation does not currently check to see that the list of ranks to ensure that there are no duplicates.

**Notes for Fortran**

All MPI routines in Fortran (except for **MPL\_WTIME** and **MPL\_WTICK**) have an additional argument  **ierr** at the end of the argument list.  **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPL\_Datatype**, **MPL\_Comm**) are of type **INTEGER** in Fortran.

**Errors**

All MPI routines (except **MPL\_Wtime** and **MPL\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPL\_Errhandler\_set**; the predefined error handler **MPL\_ERRORS\_RETURN** may

be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPL\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_GROUP**

Null group passed to function.

**MPL\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPL\_ERR\_RANK**).

**MPL\_ERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**MPL\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPL\_Recv**, **MPL\_Irecv**, **MPL\_Sendrecv**, etc.) may also be **MPL\_ANY\_SOURCE**.

**See Also**

**MPL\_Group\_free**

**Location**

`./src/context/group_incl.c`

**MPL\_Group\_intersection**

	MPL_Group_intersection
--	------------------------

**MPL\_Group\_intersection** — Produces a group as the intersection of two existing groups

**Synopsis**

`int MPI_Group_intersection ( MPI_Group group1, MPI_Group group2, MPI_Group *group_out )`

**Input Parameters**

**group1** first group (handle)  
**group2** second group (handle)

**Output Parameter**

**newgroup** intersection group (handle)

**Notes for Fortran**

All MPI routines in Fortran (except for **MPL\_WTIME** and **MPL\_WTICK**) have an additional argument  **ierr** at the end of the argument list.  **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_GROUP**

Null group passed to function.

**MPI\_ERR\_INTERN**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

See Also

`MPI_Group_free`

Location

`./src/context/group_inter.c`

<code>MPI_Group_range_excl</code>	<code>MPI_Group_range_excl</code>
-----------------------------------	-----------------------------------

**MPI\_Group\_range\_excl** — Produces a group by excluding ranges of processes from an existing group

Synopsis

```
int MPI_Group_range_excl ( MPI_Group group, int n, int ranges[][3],
                          MPI_Group *newgroup )
```

Input Parameters

- group** (handle) group (handle)
- n** number of elements in array **ranges** (integer)
- ranges** a one-dimensional array of integer triplets of the form (first rank, last rank, stride), indicating the ranks in group of processes to be excluded from the output group **newgroup**.

Output Parameter

**newgroup** new group derived from above, preserving the order in **group** (handle)

Note

Currently, each of the ranks to exclude must be a valid rank in the group and all elements must be distinct or the function is erroneous. This restriction is per the draft.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_GROUP**

Null group passed to function.

**MPI\_ERR\_INTERN**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**MPL\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

**MPL\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

See Also

`MPI_Group_free`

Location

`./src/context/group_rexcl.c`

<code>MPI_Group_range_incl</code>	<code>MPI_Group_range_incl</code>
<b>MPI_Group_range_incl</b> — Creates a new group from ranges of ranks in an existing group	

Synopsis

```
int MPI_Group_range_incl ( MPI_Group group, int n, int ranges[][3],
                          MPI_Group *newgroup )
```

Input Parameters

**group** group (handle)  
**n** number of triplets in array **ranges** (integer)  
**ranges** a one-dimensional array of integer triplets, of the form (first rank, last rank, stride) indicating ranks in **group** or processes to be included in **newgroup**

Output Parameter

**newgroup** new group derived from above, in the order defined by **ranges** (handle)

Note

This implementation does not currently check to see that the list of ranges to include are valid ranks in the group.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_GROUP**

Null group passed to function.

**MPI\_ERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

**MPI\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

See Also

**MPI\_Group\_free**

Location

`./src/context/group_rincl.c`

**MPI\_Group\_rank**

**MPI\_Group\_rank**

**MPI\_Group\_rank** — Returns the rank of this process in the given group

Synopsis

`int MPI_Group_rank ( MPI_Group group, int *rank )`

Input Parameters

**group** group (handle)

Output Parameter

**rank** rank of the calling process in group, or **MPI\_UNDEFINED** if the process is not a member (integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_GROUP**

Null group passed to function.

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

Location

`./src/context/group_rank.c`

MPI_Group_size		MPI_Group_size
MPI_Group_size — Returns the size of a group		
Synopsis		
int MPI_Group_size ( MPI_Group group, int *size )		
Input Parameters		
group	group (handle)	Output Parameter:
size	number of processes in the group (integer)	
Notes for Fortran		
All MPI routines in Fortran (except for <b>MPI_WTIME</b> and <b>MPI_WTICK</b> ) have an additional argument <b> ierr </b> at the end of the argument list. <b> ierr </b> is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the <b> call </b> statement.		
All MPI objects (e.g., <b>MPI_Datatype</b> , <b>MPI_Comm</b> ) are of type <b>INTEGER</b> in Fortran.		
Errors		
All MPI routines (except <b>MPI_Wtime</b> and <b>MPI_Wtick</b> ) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with <b>MPI_Errhandler_set</b> ; the predefined error handler <b>MPI_ERRORS_RETURN</b> may be used to cause error values to be returned. Note that MPI does <i>not</i> guarantee that an MPI program can continue past an error.		
<b>MPI_SUCCESS</b> No error; MPI routine completed successfully.		
<b>MPI_ERR_GROUP</b> Null group passed to function.		
<b>MPI_ERR_ARG</b> Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., <b>MPI_ERR_RANK</b> ).		
Location		
./src/context/group_size.c		
MPI_Group_translate_ranks		MPI_Group_translate_ranks
MPI_Group_translate_ranks — Translates the ranks of processes in one group to those in another group		

Synopsis		int MPI_Group_translate_ranks ( MPI_Group group_a, int n, int *ranks_a, MPI_Group group_b, int *ranks_b )
Input Parameters		
group1	group1 (handle)	
n	number of ranks in <b>ranks1</b> and <b>ranks2</b> arrays (integer)	
ranks1	array of zero or more valid ranks in <b>group1</b>	
group2	group2 (handle)	
Output Parameter		
ranks2	array of corresponding ranks in <b>group2</b> , <b>MPI_UNDEFINED</b> when no correspondence exists.	
Notes for Fortran		
All MPI routines in Fortran (except for <b>MPI_WTIME</b> and <b>MPI_WTICK</b> ) have an additional argument <b> ierr </b> at the end of the argument list. <b> ierr </b> is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the <b> call </b> statement.		
All MPI objects (e.g., <b>MPI_Datatype</b> , <b>MPI_Comm</b> ) are of type <b>INTEGER</b> in Fortran.		
Errors		
All MPI routines (except <b>MPI_Wtime</b> and <b>MPI_Wtick</b> ) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with <b>MPI_Errhandler_set</b> ; the predefined error handler <b>MPI_ERRORS_RETURN</b> may be used to cause error values to be returned. Note that MPI does <i>not</i> guarantee that an MPI program can continue past an error.		
<b>MPI_SUCCESS</b> No error; MPI routine completed successfully.		
<b>MPI_ERR_GROUP</b> Null group passed to function.		
<b>MPI_ERR_ARG</b> Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., <b>MPI_ERR_RANK</b> ).		
<b>MPI_ERR_RANK</b> Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive ( <b>MPI_Recv</b> , <b>MPI_Irecv</b> , <b>MPI_Sendrecv</b> , etc.) may also be <b>MPI_ANY_SOURCE</b> .		
Location		
./src/context/group_translate_ranks.c		
MPI_Group_union		MPI_Group_union
MPI_Group_union — Produces a group by combining two groups		



Synopsis

```
int MPI_Group_union ( MPI_Group group1, MPI_Group group2,
                     MPI_Group *group_out )
```

Input Parameters

group1      first group (handle)  
group2      second group (handle)

Output Parameter

newgroup    union group (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_GROUP**

Null group passed to function.

**MPLERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

See Also

MPI\_Group-free

Location

`./src/context/group_union.c`

MPI_IbSEND	MPI_IbSEND
MPI_IbSEND — Starts a nonblocking buffered send	

Synopsis

```
int MPI_IbSEND( void *buf, int count, MPI_Datatype datatype, int dest, int tag,
                MPI_Comm comm, MPI_Request *request )
```

Input Parameters

buf            initial address of send buffer (choice)  
count          number of elements in send buffer (integer)  
datatype       datatype of each send buffer element (handle)  
dest           rank of destination (integer)  
tag            message tag (integer)  
comm           communicator (handle)

Output Parameter

request        communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type  `INTEGER`  in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPLERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPLERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPLERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

**MPLERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

**MPI\_ERR\_BUFFER**

Invalid buffer pointer. Usually a null buffer where one is not valid.

**Location**

`./src/pt2pt/ibsend.c`

**MPI\_Info\_c2f**

MPI_Info_c2f	MPI_Info_c2f
--------------	--------------

**MPI\_Info\_c2f** — Translates a C info handle to a Fortran info handle

**Synopsis**

`MPI_Fint MPI_Info_c2f(MPI_Info info)`

**Input Parameters**

**info**            C info handle (integer)

**Return Value**

Fortran info handle (handle)

**Location**

`./src/misc2/info_c2f.c`

**MPI\_Info\_create**

MPI_Info_create	MPI_Info_create
-----------------	-----------------

**MPI\_Info\_create** — Creates a new info object

**Synopsis**

`int MPI_Info_create(MPI_Info *info)`

**Output Parameters**

**info**            info object (handle)

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran..

**Location**

`./src/misc2/info_create.c`

**MPI\_Info\_delete**

MPI_Info_delete	MPI_Info_delete
-----------------	-----------------

**MPI\_Info\_delete** — Deletes a (key,value) pair from info

**Synopsis**

`int MPI_Info_delete(MPI_Info info, char *key)`

**Input Parameters**

**info**            info object (handle)  
**key**            key (string)

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran..

**Location**

`./src/misc2/info_delete.c`

**MPI\_Info\_dup**

MPI_Info_dup	MPI_Info_dup
--------------	--------------

**MPI\_Info\_dup** — Returns a duplicate of the info object

**Synopsis**

`int MPI_Info_dup(MPI_Info info, MPI_Info *newinfo)`

**Input Parameters**

**info**            info object (handle)

**Output Parameters**

**newinfo**        duplicate of info object (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./src/misc2/info_dup.c`

MPI_Info_f2c	MPI_Info_f2c
--------------	--------------

**MPI\_Info\_f2c** — Translates a Fortran info handle to a C info handle

Synopsis

`MPI_Info MPI_Info_f2c(MPI_Fint info)`

Input Parameters

**info**      Fortran info handle (integer)

Return Value

C info handle (handle)

Location

`./src/misc2/info_f2c.c`

MPI_Info_free	MPI_Info_free
---------------	---------------

**MPI\_Info\_free** — Frees an info object

Synopsis

`int MPI_Info_free(MPI_Info *info)`

Input Parameters

**info**      info object (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./src/misc2/info_free.c`

MPI_Info_get	MPI_Info_get
--------------	--------------

**MPI\_Info\_get** — Retrieves the value associated with a key

Synopsis

`int MPI_Info_get(MPI_Info info, char *key, int valuelen, char *value, int *flag)`

Input Parameters

**info**      info object (handle)  
**key**      key (string)  
**valuelen** length of value argument (integer)

Output Parameters

**value**      value (string)  
**flag**      true if key defined, false if not (boolean)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./src/misc2/info_get.c`

MPI_Info_get_nkeys	MPI_Info_get_nkeys
--------------------	--------------------

**MPI\_Info\_get\_nkeys** — Returns the number of currently defined keys in info

Synopsis

int MPI\_Info\_get\_nkeys(MPI\_Info info, int \*nkeys)

Input Parameters

info info object (handle)

Output Parameters

nkeys number of defined keys (integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./src/misc2/info\_getnks.c

MPI\_Info\_get\_nthkey

MPI\_Info\_get\_nthkey

MPI\_Info\_get\_nthkey — Returns the nth defined key in info

Synopsis

int MPI\_Info\_get\_nthkey(MPI\_Info info, int n, char \*key)

Input Parameters

info info object (handle)  
n key number (integer)

Output Parameters

keys key (string)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./src/misc2/info\_getnth.c

MPI\_Info\_get\_valuelen

MPI\_Info\_get\_valuelen

MPI\_Info\_get\_valuelen — Retrieves the length of the value associated with a key

Synopsis

int MPI\_Info\_get\_valuelen(MPI\_Info info, char \*key, int \*valuelen, int \*flag)

Input Parameters

info info object (handle)  
key key (string)

Output Parameters

valuelen length of value argument (integer)  
flag true if key defined, false if not (boolean)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./src/misc2/info\_getvln.c

MPI\_Info\_set

MPI\_Info\_set

MPI\_Info\_set — Adds a (key,value) pair to info

Synopsis

int MPI\_Info\_set(MPI\_Info info, char \*key, char \*value)

Input Parameters

info info object (handle)  
key key (string)  
value value (string)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./src/misc2/info_set.c`

MPI_Init	MPI_Init
MPI_Init — Initialize the MPI execution environment	

Synopsis

`int MPI_Init(int *argc, char ***argv)`

Input Parameters

`argc` Pointer to the number of arguments  
`argv` Pointer to the argument vector

Command line arguments

MPI specifies no command-line arguments but does allow an MPI implementation to make use of them.

- mpiqueue** print out the state of the message queues when `MPI_FINALIZE` is called. All processors print; the output may be hard to decipher. This is intended as a debugging aid.
- mpiversion** print out the version of the implementation (*not* of MPI), including the arguments that were used with configure.
- mpinice nn** Increments the nice value by `nn` (lowering the priority of the program by `nn`). `nn` must be positive (except for root). Not all systems support this argument; those that do not will ignore it.
- mpedbg** Start a debugger in an xterm window if there is an error (either detected by MPI or a normally fatal signal). This works only if MPICH was configured with `-mpedbg`. **CURRENTLY DISABLED**. If you have `TotalView`, `-mpicdiv` or `mpirun -tv` will give you a better environment anyway.
- mpimem** If MPICH was built with `-DMPID_DEBUG_MEM`, this checks all malloc and free operations (internal to MPI) for signs of injury to the memory allocation areas.
- mpidb options** Activate various debugging options. Some require that MPICH have been built with special options. These are intended for debugging MPICH, not for debugging user programs. The available options include:
  - `mem` - Enable dynamic memory tracing of internal MPI objects
  - `memall` - Generate output of all memory allocation/deallocation
  - `ptr` - Enable tracing of internal MPI pointer conversions
  - `rank n` - Limit subsequent `-mpidb` options to on the process with

the specified rank in `MPI_COMM_WORLD`. A rank of `-1` selects all of `MPI_COMM_WORLD`.

`ref` - Trace use of internal MPI objects

`reffile filename` - Trace use of internal MPI objects with output to the indicated file

`trace` - Trace routine calls

Notes

Note that the Fortran binding for this routine has only the error return argument (`MPI_INIT(ierr)`)

Because the Fortran and C versions of `MPI_Init` are different, there is a restriction on who can call `MPI_Init`. The version (Fortran or C) must match the main program. That is, if the main program is in C, then the C version of `MPI_Init` must be called. If the main program is in Fortran, the Fortran version must be called.

On exit from this routine, all processes will have a copy of the argument list. This is *not required* by the MPI standard, and truly portable codes should not rely on it. This is provided as a service by this implementation (an MPI implementation is allowed to distribute the command line arguments but is not required to).

Command line arguments are not provided to Fortran programs. More precisely, non-standard Fortran routines such as `getarg` and `iargc` have undefined behavior in MPI and in this implementation.

The MPI standard does not say what a program can do before an `MPI_INIT` or after an `MPI_FINALIZE`. In the MPICH implementation, you should do as little as possible. In particular, avoid anything that changes the external state of the program, such as opening files, reading standard input or writing to standard output.

Signals used

The MPI standard requires that all signals used be documented. The MPICH implementation itself uses no signals, but some of the software that MPICH relies on may use some signals. The list below is partial and should be independently checked if you (and any package that you use) depend on particular signals.

IBM POE/MPL for SP2

`SIGHUP`, `SIGINT`, `SIGQUIT`, `SIGFPE`, `SIGSEGV`, `SIGPIPE`, `SIGALRM`, `SIGTERM`, `SIGIO`

-mpedbg switch

`SIGQUIT`, `SIGILL`, `SIGFPE`, `SIGBUS`, `SIGSEGV`, `SIGSYS`

Meiko CS2

`SIGUSR2`

ch\_p4 device

`SIGUSRI`

The `ch_p4` device also catches `SIGINT`, `SIGFPE`, `SIGBUS`, and `SIGSEGV`; this helps the p4 device (and MPICH) more gracefully abort a failed program.

**Intel Paragon (ch\_nx and nx device)**  
SIGUSR2

**Shared Memory (ch\_shmem device)**

SIGCHLD

Note that if you are using software that needs the same signals, you may find that there is no way to use that software with the MPI implementation. The signals that cause the most trouble for applications include SIGIO, SIGALRM, and SIGPIPE. For example, using SIGIO and SIGPIPE may prevent X11 routines from working.

**Errors**

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_OTHER**

This error class is associated with an error code that indicates that an attempt was made to call `MPI_INIT` a second time. `MPI_INIT` may only be called once in a program.

**Location**

`./src/env/init.c`

MPI_Init_thread	MPI_Init_thread
MPI_Init_thread — Initialize the MPI execution environment	

**Synopsis**

`int MPI_Init_thread(int *argc, char ***argv, int required, int *provided )`

**Input Parameters**

- `argc` Pointer to the number of arguments
- `argv` Pointer to the argument vector
- `required` Level of desired thread support

**Output Parameter**

- `provided` Level of provided thread support

**Command line arguments**

MPI specifies no command-line arguments but does allow an MPI implementation to make use of them. See `MPI_INIT` for a description of the command line arguments supported by `MPI_INIT` and `MPI_INIT_THREAD`.

**Notes**

Note that the Fortran binding for this routine does not have the `argc` and `argv` arguments. (`MPI_INIT_THREAD(required, provided, ierror)`)  
Currently, MPICH places the same restrictions on `MPI_INIT_THREAD` as on `MPI_INIT` (see the `MPI_INIT` man page). When MPICH fully supports MPI-2, this restriction will be removed (as required by the MPI-2 standard).

**Signals used**

The MPI standard requires that all signals used be documented. The MPICH implementation itself uses no signals, but some of the software that MPICH relies on may use some signals. The list below is partial and should be independently checked if you (and any package that you use) depend on particular signals.

**IBM POE/MPL for SP2**

SIGHUP, SIGINT, SIGQUIT, SIGFPE, SIGSEGV, SIGPIPE, SIGALRM, SIGTERM, SIGIO

**-mpedbg switch**

SIGQUIT, SIGILL, SIGFPE, SIGBUS, SIGSEGV, SIGSYS

**Meiko CS2**

SIGUSR2

**ch\_p4 device**

SIGUSR1

The `ch_p4` device also catches SIGINT, SIGFPE, SIGBUS, and SIGSEGV; this helps the `p4` device (and MPICH) more gracefully abort a failed program.

**Intel Paragon (ch\_nx and nx device)**

SIGUSR2

**Shared Memory (ch\_shmem device)**

SIGCHLD

Note that if you are using software that needs the same signals, you may find that there is no way to use that software with the MPI implementation. The signals that cause the most trouble for applications include SIGIO, SIGALRM, and SIGPIPE. For example, using SIGIO and SIGPIPE may prevent X11 routines from working.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRHBS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_OTHER

This error class is associated with an error code that indicates that an attempt was made to call `MPI_INIT` a second time. `MPI_INIT` may only be called once in a program.

Location

`./src/env/initthread.c`

MPI_Initialized	MPI_Initialized
MPI_Initialized — Indicates whether MPI_Init has been called.	

Synopsis

`int MPI_Initialized( int *flag )`

Output Parameter

flag Flag is true if `MPI_Init` has been called and false otherwise.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./src/env/initialize.c`

MPI_Intercomm_create	MPI_Intercomm_create
MPI_Intercomm_create — Creates an intercommunicator from two intracommunicators	

Synopsis

```
int MPI_Intercomm_create ( MPI_Comm local_comm, int local_leader,
                           MPI_Comm peer_comm, int remote_leader, int tag,
                           MPI_Comm *comm_out )
```

Input Paramters

`local_comm` Local (intra)communicator  
`local_leader` Rank in `local_comm` of leader (often 0)  
`peer_comm` Remote communicator  
`remote_leader` Rank in `peer_comm` of remote leader (often 0)

`tag` Message tag to use in constructing intercommunicator; if multiple `MPI_Intercomm_creates` are being made, they should use different tags (more precisely, ensure that the local and remote leaders are using different tags for each `MPI_intercomm_create`).

Output Parameter

`comm_out` Created intercommunicator

Notes

The MPI 1.1 Standard contains two mutually exclusive comments on the input intracommunicators. One says that their respective groups must be disjoint; the other that the leaders can be the same process. After some discussion by the MPI Forum, it has been decided that the groups must be disjoint. Note that the *reason* given for this in the standard is *not* the reason for this choice; rather, the *other* operations on intercommunicators (like `MPI_Intercomm_merge`) do not make sense if the groups are not disjoint.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Algorithm

- 1) Allocate a send context, an intercoll context, and an intra-coll context
- 2) Send "send\_context" and rank\_to\_grank list from local comm group if I'm the local\_leader.
- 3) If I'm the local leader, then wait on the posted sends and receives to complete. Post the receive for the remote group information and wait for it to complete.
- 4) Broadcast information received from the remote leader.
  - 5) Create the inter\_communicator from the information we now have.

**An inter**      communicator ends up with three levels of communicators. The inter-communicator returned to the user, a "collective" inter-communicator that can be used for safe communications between local & remote groups, and a collective intra-communicator that can be used to allocate new contexts during the merge and dup operations.

For the resulting inter-communicator, `comm_out`

```
comm_out = inter-communicator
comm_out->comm_coll = "collective" inter-communicator
comm_out->comm_coll->comm_coll = safe collective intra-communicator
```

**Errors**

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPL\_ERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

**MPL\_ERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**MPL\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

**See Also**

`MPI_Intercomm_merge`, `MPI_Comm_free`, `MPI_Comm_remote_group`, `MPI_Comm_remote_size`

**Location**

`./src/context/ic_create.c`

---

`MPI_Intercomm_merge`

`MPI_Intercomm_merge`

`MPI_Intercomm_merge` — Creates an intra-communicator from an intercommunicator

**Synopsis**

```
int MPI_Intercomm_merge ( MPI_Comm comm, int high, MPI_Comm *comm_out )
```

**Input Parameters**

**comm**      Intercommunicator  
**high**      Used to order the groups of the two intracommunicators within comm when creating the new communicator.

**Output Parameter**

**comm\_out**      Created intracommunicator

**Notes for Fortran**

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

**Algorithm**

- 1) Allocate two contexts
- 2) Local and remote group leaders swap high values
- 3) Determine the high value.
- 4) Merge the two groups and make the intra-communicator

**Errors**

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPL\_ERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**See Also**

`MPI_Intercomm_create`, `MPI_Comm_free`

**Location**

`./src/context/ic_merge.c`



MPI_Iprobe	MPI_Iprobe
------------	------------

MPI\_Iprobe — Nonblocking test for a message

Synopsis

```
int MPI_Iprobe( int source, int tag, MPI_Comm comm, int *flag,
               MPI_Status *status )
```

Input Parameters

source      source rank, or MPI\_ANY\_SOURCE (integer)  
tag        tag value or MPI\_ANY\_TAG (integer)  
comm       communicator (handle)

Output Parameter

flag        (logical)  
status      status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call

(not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_TAG**. The largest tag value is available through the attribute **MPI\_TAG\_UB**.

**MPI\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

Location

./src/pt2pt/iprobe.c

MPI_Irecv	MPI_Irecv
-----------	-----------

MPI\_Irecv — Begins a nonblocking receive

Synopsis

```
int MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int source,
               int tag, MPI_Comm comm, MPI_Request *request )
```

Input Parameters

buf        initial address of receive buffer (choice)  
count      number of elements in receive buffer (integer)  
datatype   datatype of each receive buffer element (handle)  
source     rank of source (integer)  
tag        message tag (integer)  
comm       communicator (handle)

Output Parameter

request    communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./src/pt2pt/irecv.c

MPI_Irsend	MPI_Irsend
------------	------------

MPI\_Irsend — Starts a nonblocking ready send

Synopsis

```
int MPI_Irsend( void *buf, int count, MPI_Datatype datatype, int dest,
                int tag, MPI_Comm comm, MPI_Request *request )
```

### Input Parameters

**buf** initial address of send buffer (choice)  
**count** number of elements in send buffer (integer)  
**datatype** datatype of each send buffer element (handle)  
**dest** rank of destination (integer)  
**tag** message tag (integer)  
**comm** communicator (handle) **Output Parameter:**  
**request** communication request (handle)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS** No error; MPI routine completed successfully.

**MPLERR\_COMM** Invalid communicator. A common error is to use a null communicator in a call

(not even allowed in **MPI\_Comm\_rank**).

**MPLERR\_COUNT** Invalid count argument. Count arguments must be non-negative; a count of zero is

often valid.

**MPLERR\_TYPE** Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

**MPLERR\_TAG** Invalid tag argument. Tags must be non-negative; tags in a receive (**MPI\_Recv**,

**MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_TAG**. The largest tag value is available through the attribute **MPI\_TAG\_UB**.

**MPLERR\_RANK** Invalid source or destination rank. Ranks must be between zero and the size of

the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

**MPLERR\_INTERR** This error is returned when some part of the MPICH implementation is unable to

acquire memory.

### Location

`./src/pt2pt/irsend.c`

---

**MPI\_Isend**

---

**MPI\_Isend** — Begins a nonblocking send

### Synopsis

```
int MPI_Isend( void *buf, int count, MPI_Datatype datatype, int dest, int tag,
               MPI_Comm comm, MPI_Request *request )
```

### Input Parameters

**buf** initial address of send buffer (choice)  
**count** number of elements in send buffer (integer)  
**datatype** datatype of each send buffer element (handle)  
**dest** rank of destination (integer)  
**tag** message tag (integer)  
**comm** communicator (handle)

### Output Parameter

**request** communication request (handle)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS** No error; MPI routine completed successfully.

**MPLERR\_COMM** Invalid communicator. A common error is to use a null communicator in a call

(not even allowed in **MPI\_Comm\_rank**).

**MPLERR\_COUNT** Invalid count argument. Count arguments must be non-negative; a count of zero is

often valid.

**MPLERR\_TYPE** Invalid datatype argument. May be an uncommitted MPI\_Datatype (see

**MPI\_Type\_commit**).

**MPLERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_TAG**. The largest tag value is available through the the attribute **MPI\_TAG\_UB**.

**MPI\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

**MPI\_ERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**Location**

`./src/pr2pt/issend.c`

<b>MPI_Issend</b>	<b>MPI_Issend</b>
-------------------	-------------------

**MPI\_Issend** — Starts a nonblocking synchronous send

**Synopsis**

```
int MPI_Issend( void *buf, int count, MPI_Datatype datatype, int dest,
               int tag, MPI_Comm comm, MPI_Request *request )
```

**Input Parameters**

**buf**            initial address of send buffer (choice)  
**count**        number of elements in send buffer (integer)  
**datatype**     datatype of each send buffer element (handle)  
**dest**         rank of destination (integer)  
**tag**          message tag (integer)  
**comm**         communicator (handle)

**Output Parameter**

**request**       communication request (handle)

**Notes for Fortran**

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr** at the end of the argument list.  **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

**Errors**

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler

may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see

**MPI\_Type\_commit**).

**MPI\_ERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_TAG**. The largest tag value is available through the the attribute **MPI\_TAG\_UB**.

**MPI\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**,

**MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

**MPI\_ERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**Location**

`./src/pr2pt/issend.c`

<b>MPI_Keyval_create</b>	<b>MPI_Keyval_create</b>
--------------------------	--------------------------

**MPI\_Keyval\_create** — Generates a new attribute key

**Synopsis**

```
int MPI_Keyval_create (
    MPI_Copy_function *copy_fn,
    MPI_Delete_function *delete_fn,
    int *keyval,
    void *extra_state )
```

**Input Parameters**

**copy\_fn**       Copy callback function for **keyval**  
**delete\_fn**     Delete callback function for **keyval**  
**extra\_state**   Extra state for callback functions

**Output Parameter**

**keyval**        key value for future access (integer)

Notes

Key values are global (available for any and all communicators). There are subtle differences between C and Fortran that require that the `copy_fn` be written in the same language that `MPI_Keyval_create` is called from. This should not be a problem for most users; only programmers using both Fortran and C in the same program need to be sure that they follow this rule.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.  
**MPI\_SUCCESS** No error; MPI routine completed successfully.  
**MPI\_ERR\_INTERR** Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).  
**MPI\_ERR\_ARG** This error is returned when some part of the MPICH implementation is unable to acquire memory.

Location

`./src/context/keyvalcreate.c`

MPI_Keyval_free	MPI_Keyval_free
MPI_Keyval_free — Frees attribute key for communicator cache attribute	

Synopsis

`int MPI_Keyval_free ( int *keyval )`

Input Parameter

keyval      Frees the integer key value (integer)

Note

Key values are global (they can be used with any and all communicators)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.  
**MPI\_SUCCESS** No error; MPI routine completed successfully.  
**MPI\_ERR\_ARG** Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).  
**MPI\_ERR\_ARG** This error class is associated with an error code that indicates that an attempt was made to free one of the permanent keys.

See Also

`MPI_Keyval_create`

Location

`./src/context/keyval_free.c`

MPI_NULL_COPY_FN	MPI_NULL_COPY_FN
MPI_NULL_COPY_FN — A function to not copy attributes	

Notes

See discussion of `MPI_Keyval_create` for the use of this function.

Location

`./src/context/null_copyfn.c`

MPI_NULL_DELETE_FN	MPI_NULL_DELETE_FN
MPI_NULL_DELETE_FN — A function to not delete attributes	

Input Parameters

comm	Communicator
keyval	Key value
attr	attribute
extra_state	User-defined state to give user functions

Notes

See discussion of **MPI\_Keyval\_create** for the use of this function.

Location

`./src/context/null_del_fn.c`

MPI_Op_create	MPI_Op_create
MPI_Op_create — Creates a user-defined combination function handle	

Synopsis

```
int MPI_Op_create(  
    MPI_User_function *function,  
    int commute,  
    MPI_Op *op )
```

Input Parameters

**function**      user defined function (function)  
**commute**        true if commutative; false otherwise.

Output Parameter

**op**             operation (handle)

Notes on the user function

The calling list for the user function type is

```
typedef void (MPI_User_function) ( void * a,  
    void * b, int * len, MPI_Datatype * );
```

where the operation is `b[i] = a[i] op b[i]`, for `i=0,...,len-1`. A pointer to the datatype given to the MPI collective computation routine (i.e., `MPI_Reduce`, `MPI_Allreduce`, `MPI_Scan`, or `MPI_Reduce_scatter`) is also passed to the user-specified routine.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Notes on collective operations

The reduction functions (`MPI_Op`) do not return an error value. As a result, if the functions detect an error, all they can do is either call `MPI_Abort` or silently skip the problem. Thus, if you change the error handler from `MPI_ERRORS_ARE_FATAL` to something else, for example, `MPI_ERRORS_RETURN`, then no error may be indicated.

The reason for this is the performance problems in ensuring that all collective routines return the same error value.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_INTERR

This error is returned when some part of the MPICH implementation is unable to acquire memory.

See Also

`MPI_Op_free`

Location

`./src/coll/opcreate.c`

MPI_Op_free	MPI_Op_free
MPI_Op_free — Frees a user-defined combination function handle	

Synopsis

```
int MPI_Op_free( MPI_Op *op )
```

Input Parameter

op                    operation (handle)

Notes

op is set to MPI\_OP\_NULL on exit.

Null Handles

The MPI 1.1 specification, in the section on opaque objects, explicitly

disallows freeing a null communicator. The text from the standard is

A null handle argument is an erroneous IN argument in MPI calls, unless an exception is explicitly stated in the text that defines the function. Such exception is allowed for handles to request objects in Wait and Test calls (sections Communication Completion and Multiple Completions ). Otherwise, a null handle can only be passed to a function that allocates a new object and returns a reference to it in the handle.

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Errors

All MPI routines (except MPI\_Wtime and MPI\_Wtick) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with MPI\_Errhandler\_set; the predefined error handler MPI\_ERRORS\_RETURN may be used to cause error values to be returned. Note that MPI does not guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., MPI\_ERR\_RANK).

MPI\_ERR\_ARG

Invalid argument; the error code associated with this error indicates an attempt to free an MPI permanent operation (e.g., MPI\_SUM). \*N/ \*N

MPI\_ERR\_ARG

Invalid argument; the error code associated with this error indicates an attempt to free or change an MPI permanent keyval (e.g., MPI\_TAG\_UB). \*N/ \*N

MPI\_ERR\_UNKNOWN

Unknown error. You should never see this. If you do, report it to mpi-bugs@ncs.anl.gov.

See Also

MPI\_Op\_create

Location

./src/coll/opfree.c

MPI\_Pack

MPI\_Pack — Packs a datatype into contiguous memory

Synopsis

int MPI\_Pack ( void \*inbuf, int incount, MPI\_Datatype datatype, void \*outbuf, int outcount, int \*position, MPI\_Comm comm )

Input Parameters

inbuf                input buffer start (choice)  
incount             number of input data items (integer)  
datatype            datatype of each input data item (handle)  
outcount            output buffer size, in bytes (integer)  
position            current position in buffer, in bytes (integer)  
comm                communicator for packed message (handle)

Output Parameter

outbuf              output buffer start (choice)

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Errors

All MPI routines (except MPI\_Wtime and MPI\_Wtick) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with MPI\_Errhandler\_set; the predefined error handler MPI\_ERRORS\_RETURN may be used to cause error values to be returned. Note that MPI does not guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPI\_ERR\_TYPE**  
Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPI\_ERR\_COUNT**  
Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_ARG**  
Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

See Also

`MPI_Unpack`, `MPI_Pack_size`

Location

`./src/pt2pt/pack.c`

<code>MPI_Pack_size</code>	<code>MPI_Pack_size</code>
<hr/>	
<code>MPI_Pack_size</code> — Returns the upper bound on the amount of space needed to pack a message	

Synopsis

```
int MPI_Pack_size ( int incout, MPI_Datatype datatype, MPI_Comm comm,
                   int *size )
```

Input Parameters

`incout` count argument to packing call (integer)  
`datatype` datatype argument to packing call (handle)  
`comm` communicator argument to packing call (handle)

Output Parameter

`size` upper bound on size of packed message, in bytes (integer)

Notes

The MPI standard document describes this in terms of `MPI_Pack`, but it applies to both `MPI_Pack` and `MPI_Unpack`. That is, the value `size` is the maximum that is needed by either `MPI_Pack` or `MPI_Unpack`.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**  
No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**  
Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPI\_ERR\_TYPE**  
Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPI\_ERR\_ARG**  
Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/pt2pt/pack_size.c`

<code>MPI_Pcontrol</code>	<code>MPI_Pcontrol</code>
<hr/>	
<code>MPI_Pcontrol</code> — Controls profiling	

Synopsis

```
int MPI_Pcontrol( int level )
```

Input Parameters

`level` Profiling level

Notes

This routine provides a common interface for profiling control. The interpretation of `level` and any other arguments is left to the profiling library.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./src/profile/pcontrol.c

MPI_Probe	MPI_Probe
MPI_Probe — Blocking test for a message	

Synopsis

int MPI\_Probe( int source, int tag, MPI\_Comm comm, MPI\_Status \*status )

Input Parameters

source      source rank, or **MPI\_ANY\_SOURCE** (integer)  
tag        tag value or **MPI\_ANY\_TAG** (integer)  
comm      communicator (handle)

Output Parameter

status      status object (Status)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.  
**MPI\_SUCCESS**      No error; MPI routine completed successfully.  
**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPI\_ERR\_TAG

Invalid tag argument. Tags must be non-negative; tags in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_TAG**. The largest tag value is available through the the attribute **MPI\_TAG\_UB**.

MPI\_ERR\_RANK

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

Location

./src/pt2pt/probe.c

MPI_Recv	MPI_Recv
MPI_Recv — Basic receive	

Synopsis

int MPI\_Recv( void \*buf, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm comm, MPI\_Status \*status )

Output Parameters

buf        initial address of receive buffer (choice)  
status     status object (Status)

Input Parameters

count      maximum number of elements in receive buffer (integer)  
datatype   datatype of each receive buffer element (handle)  
source     rank of source (integer)  
tag        message tag (integer)  
comm      communicator (handle)

Notes

The **count** argument indicates the maximum length of a message; the actual number can be determined with **MPI\_Get\_count**.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  **call**  statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.



Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPI\_ERR\_TAG

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

MPI\_ERR\_RANK

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

Location

`./src/pt2pt/recv.c`

MPI\_Recv\_init MPI\_Recv\_init

`MPI_Recv_init` — Builds a handle for a receive

Synopsis

```
int MPI_Recv_init( void *buf, int count, MPI_Datatype datatype, int source,
                  int tag, MPI_Comm comm, MPI_Request *request )
```

Input Parameters

`buf` initial address of receive buffer (choice)  
`count` number of elements received (integer)  
`datatype` type of each element (handle)  
`source` rank of source or `MPI_ANY_SOURCE` (integer)  
`tag` message tag or `MPI_ANY_TAG` (integer)  
`comm` communicator (handle)

Output Parameter

`request` communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

MPI\_ERR\_RANK

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

MPI\_ERR\_TAG

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPI\_ERR\_INTERN

This error is returned when some part of the MPICH implementation is unable to acquire memory.

See Also

`MPI_Start`, `MPI_Request_free`

Location

`./src/pt2pt/create_recv.c`

MPI_Reduce	MPI_Reduce
MPI_Reduce — Reduces values on all processes to a single value	

Synopsis

```
int MPI_Reduce ( void *sendbuf, void *recvbuf, int count,  
MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm )
```

Input Parameters

- sendbuf** address of send buffer (choice)
- count** number of elements in send buffer (integer)
- datatype** data type of elements of send buffer (handle)
- op** reduce operation (handle)
- root** rank of root process (integer)
- comm** communicator (handle)

Output Parameter

- recvbuf** address of receive buffer (choice, significant only at root)

Algorithm

This implementation currently uses a simple tree algorithm.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Notes on collective operations

The reduction functions (**MPI\_Op**) do not return an error value. As a result, if the functions detect an error, all they can do is either call **MPI\_Abort** or silently skip the problem. Thus, if you change the error handler from **MPI\_ERRORS\_ARE\_FATAL** to something else, for example, **MPI\_ERRORS\_RETURN**, then no error may be indicated.  
The reason for this is the performance problems in ensuring that all collective routines return the same error value.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may

be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see **MPI\_Type\_commit**).

MPI\_ERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

MPI\_ERR\_BUFFER

This error class is associated with an error code that indicates that two buffer arguments are *aliased*; that is, the describe overlapping storage (often the exact same storage). This is prohibited in MPI (because it is prohibited by the Fortran standard, and rather than have a separate case for C and Fortran, the MPI Forum adopted the more restrictive requirements of Fortran).

Location

`./src/coll/reduce.c`

MPI_Reduce_scatter	MPI_Reduce_scatter
MPI_Reduce_scatter — Combines values and scatters the results	

Synopsis

```
int MPI_Reduce_scatter ( void *sendbuf, void *recvbuf, int *recvcnts,  
MPI_Datatype datatype, MPI_Op op, MPI_Comm comm )
```

Input Parameters

- sendbuf** starting address of send buffer (choice)
- recvcounts** integer array specifying the number of elements in result distributed to each process. Array must be identical on all calling processes.
- datatype** data type of elements of input buffer (handle)
- op** operation (handle)
- comm** communicator (handle)

Output Parameter

- recvbuf** starting address of receive buffer (choice)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Notes on collective operations

The reduction functions (`MPI_Op`) do not return an error value. As a result, if the functions detect an error, all they can do is either call `MPI_Abort` or silently skip the problem. Thus, if you change the error handler from `MPI_ERRORS_ARE_FATAL` to something else, for example, `MPI_ERRORS_RETURN`, then no error may be indicated.

The reason for this is the performance problems in ensuring that all collective routines return the same error value.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

MPI\_ERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

MPI\_ERR\_OP

Invalid operation. MPI operations (objects of type `MPI_Op`) must either be one of the predefined operations (e.g., `MPI_SUM`) or created with `MPI_Op_create`.

MPI\_ERR\_BUFFER

This error class is associated with an error code that indicates that two buffer arguments are *aliased*; that is, the describe overlapping storage (often the exact same storage). This is prohibited in MPI (because it is prohibited by the Fortran standard, and rather than have a separate case for C and Fortran, the MPI Forum adopted the more restrictive requirements of Fortran).

Location

`./src/coll/red_scatter.c`

MPI\_Request\_c2f

`MPI_Request_c2f` — Convert a C request to a Fortran request

Synopsis

```
MPI_Fint MPI_Request_c2f( c_request )
MPI_Request c_request;
```

Input Parameters

`c_request` Request value in C (handle)

Output Value

`f_request` Status value in Fortran (Integer)

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/misc2/requestc2f.c`

MPI\_Request\_free

`MPI_Request_free` — Frees a communication request object

Synopsis

```
int MPI_Request_free( MPI_Request *request )
```

Input Parameter

`request` communication request (handle)

Notes

This routine is normally used to free persistent requests created with either `MPI_Recv_init` or `MPI_Send_init` and friends. However, it can be used to free a request created with `MPI_Irecv` or `MPI_Isend` and friends; in that case the use can not use the `test/wait` routines on the request. It is permitted to free an active request. However, once freed, you can not use the request in a wait or test routine (e.g., `MPI_Wait`).

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_REQUEST**

Invalid `MPI_Request`. Either null or, in the case of a `MPI_Start` or `MPI_Startall`, not a persistent request.

**MPLERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

See Also

also: `MPI_Isend`, `MPI_Irecv`, `MPI_Isend`, `MPI_Ibsend`, `MPI_Isend`, `MPI_Isend`, `MPI_Recv_init`, `MPI_Send_init`, `MPI_Ssend_init`, `MPI_Rsend_init`, `MPI_Test`, `MPI_Waitall`, `MPL_Waitany`, `MPI_Waitsome`, `MPI_Testall`, `MPI_Testany`, `MPI_Testsome`

Location

`./src/pr2pt/commreq_free.c`

MPI_Rsend	MPI_Rsend
MPI_Rsend — Basic ready send	

Synopsis

```
int MPI_Rsend( void *buf, int count, MPI_Datatype datatype, int dest,
               int tag, MPI_Comm comm )
```

Input Parameters

**buf** initial address of send buffer (choice)  
**count** number of elements in send buffer (nonnegative integer)  
**datatype** datatype of each send buffer element (handle)  
**dest** rank of destination (integer)  
**tag** message tag (integer)  
**comm** communicator (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPLERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPLERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPLERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Ssendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

**MPLERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Ssendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

Location

`./src/pr2pt/rsend.c`

MPI_Rsend_init	MPI_Rsend_init
MPI_Rsend_init — Builds a handle for a ready send	

Synopsis

```
int MPI_Rsend_init( void *buf, int count, MPI_Datatype datatype, int dest,
                   int tag, MPI_Comm comm, MPI_Request *request )
```

Input Parameters

**buf** initial address of send buffer (choice)  
**count** number of elements sent (integer)  
**datatype** type of each element (handle)  
**dest** rank of destination (integer)  
**tag** message tag (integer)  
**comm** communicator (handle)

Output Parameter

**request** communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

**MPI\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

**MPI\_ERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_TAG**. The largest tag value is available through the the attribute **MPI\_TAG\_UB**.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPI\_ERR\_INTERR

This error is returned when some part of the MPICH implementation is unable to acquire memory.

See Also

**MPI\_Start**, **MPI\_Request\_free**, **MPI\_Send\_init**

Location

`./src/pt2pt/rsend_init.c`

MPI\_Scan

**MPI\_Scan** — Computes the scan (partial reductions) of data on a collection of processes

Synopsis

```
int MPI_Scan ( void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype,
               MPI_Op op, MPI_Comm comm )
```

Input Parameters

**sendbuf** starting address of send buffer (choice)  
**count** number of elements in input buffer (integer)  
**datatype** data type of elements of input buffer (handle)  
**op** operation (handle)  
**comm** communicator (handle)

Output Parameter

**recvbuf** starting address of receive buffer (choice)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Notes on collective operations

The reduction functions (**MPI\_Op**) do not return an error value. As a result, if the functions detect an error, all they can do is either call **MPI\_Abort** or silently skip the problem. Thus, if you change the error handler from **MPI\_ERRORS\_ARE\_FATAL** to something else, for example, **MPI\_ERRORS\_RETURN**, then no error may be indicated.

The reason for this is the performance problems in ensuring that all collective routines return the same error value.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

MPI\_ERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

MPI\_ERR\_BUFFER

This error class is associated with an error code that indicates that two buffer arguments are *aliased*; that is, the describe overlapping storage (often the exact same storage). This is prohibited in MPI (because it is prohibited by the Fortran standard, and rather than have a separate case for C and Fortran, the MPI Forum adopted the more restrictive requirements of Fortran).

Location

./src/coll/scan.c

MPI_Scatter	MPI_Scatter
MPI_Scatter — Sends data from one task to all other tasks in a group	

Synopsis

```
int MPI_Scatter (
    void *sendbuf,
    int sendcnt,
    MPI_Datatype sendtype,
    void *recvbuf,
    int recvcnt,
    MPI_Datatype recvttype,
    int root,
    MPI_Comm comm )
```

Input Parameters

**sendbuf**      address of send buffer (choice, significant only at **root**)  
**sendcount**    number of elements sent to each process (integer, significant only at **root**)

**sendtype**      data type of send buffer elements (significant only at **root**) (handle)  
**recvcount**    number of elements in receive buffer (integer)  
**recvtype**      data type of receive buffer elements (handle)  
**root**           rank of sending process (integer)  
**comm**          communicator (handle)

Output Parameter

**recvbuf**       address of receive buffer (choice)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_Wtime** and **MPI\_Wtick**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_COMM

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

MPI\_ERR\_BUFFER

Invalid buffer pointer. Usually a null buffer where one is not valid.

Location

./src/coll/scatter.c

MPI_Scatterv	MPI_Scatterv
MPI_Scatterv — Scatters a buffer in parts to all tasks in a group	

Synopsis

```
int MPI_Scatterv (
    void *sendbuf,
```

```

int *sendents,
int *displs,
MPI_Datatype sendtype,
void *recvbuf,
int recvcnt,
MPI_Datatype recvtype,
int root,
MPI_Comm comm )

```

### Input Parameters

**sendbuf** address of send buffer (choice, significant only at **root**)

**sendcounts** integer array (of length group size) specifying the number of elements to send to each processor

**displs** integer array (of length group size). Entry **i** specifies the displacement (relative to sendbuf from which to take the outgoing data to process **i**)

**sendtype** data type of send buffer elements (handle)

**recvcount** number of elements in receive buffer (integer)

**recvtype** data type of receive buffer elements (handle)

**root** rank of sending process (integer)

**comm** communicator (handle)

### Output Parameter

**recvbuf** address of receive buffer (choice)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPLERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPLERR\_TYPE**

Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see **MPI\_Type\_commit**).

**MPLERR\_BUFFER**

Invalid buffer pointer. Usually a null buffer where one is not valid.

### Location

`./src/coll/scatterv.c`

---

**MPI\_Send** **MPI\_Send**

---

**MPI\_Send** — Performs a basic send

### Synopsis

```

int MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm )

```

### Input Parameters

**buf** initial address of send buffer (choice)

**count** number of elements in send buffer (nonnegative integer)

**datatype** datatype of each send buffer element (handle)

**dest** rank of destination (integer)

**tag** message tag (integer)

**comm** communicator (handle)

### Notes

This routine may block until the message is received.

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPI\_ERR\_COUNT**  
Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_TYPE**  
Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPI\_ERR\_TAG**  
Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the the attribute `MPI_TAG_UB`.

**MPI\_ERR\_RANK**  
Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

See Also

`MPI_Isend`, `MPI_Bsend`

Location

`./src/pr2pt/send.c`

<code>MPI_Send_init</code>	<code>MPI_Send_init</code>
<code>MPI_Send_init</code> — Builds a handle for a standard send	

Synopsis

```
int MPI_Send_init( void *buf2, int count, MPI_Datatype datatype, int dest,  
                  int tag, MPI_Comm comm, MPI_Request *request )
```

Input Parameters

<b>buf</b>	initial address of send buffer (choice)
<b>count</b>	number of elements sent (integer)
<b>datatype</b>	type of each element (handle)
<b>dest</b>	rank of destination (integer)
<b>tag</b>	message tag (integer)
<b>comm</b>	communicator (handle)
<b>request</b>	communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**  
No error; MPI routine completed successfully.

**MPI\_ERR\_COUNT**  
Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_TYPE**  
Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPI\_ERR\_RANK**  
Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

**MPI\_ERR\_TAG**  
Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the the attribute `MPI_TAG_UB`.

**MPI\_ERR\_COMM**  
Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPI\_ERR\_INTERR**  
This error is returned when some part of the MPICH implementation is unable to acquire memory.

See Also

`MPI_Start`, `MPI_Startall`, `MPI_Request_free`

Location

`./src/pr2pt/create_send.c`

<code>MPI_Sendrecv</code>	<code>MPI_Sendrecv</code>
<code>MPI_Sendrecv</code> — Sends and receives a message	

Synopsis

```
int MPI_Sendrecv( void *sendbuf, int sendcount, MPI_Datatype sendtype,  
                  int dest, int sendtag,  
                  void *recvbuf, int recvcount, MPI_Datatype recvtype,  
                  int source, int recvtag, MPI_Comm comm, MPI_Status *status )
```



### Input Parameters

**sendbuf** initial address of send buffer (choice)  
**sendcount** number of elements in send buffer (integer)  
**sendtype** type of elements in send buffer (handle)  
**dest** rank of destination (integer)  
**sendtag** send tag (integer)  
**recvcount** number of elements in receive buffer (integer)  
**recvtype** type of elements in receive buffer (handle)  
**source** rank of source (integer)  
**recvtag** receive tag (integer)  
**comm** communicator (handle)

### Output Parameters

**recvbuf** initial address of receive buffer (choice)  
**status** status object (Status). This refers to the receive operation.

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS** No error; MPI routine completed successfully.

**MPLERR\_COMM** Invalid communicator. A common error is to use a null communicator in a call

(not even allowed in **MPI\_Comm\_rank**).

**MPLERR\_COUNT** Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPLERR\_TYPE** Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see **MPI\_Type\_commit**).

**MPLERR\_TAG** Invalid tag argument. Tags must be non-negative; tags in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_TAG**. The largest tag value is available through the attribute **MPI\_TAG\_UB**.

**MPLERR\_RANK** Invalid source or destination rank. Ranks must be between zero and the size of

the communicator minus one; ranks in a receive (**MPI\_Recv**, **MPI\_Irecv**, **MPI\_Sendrecv**, etc.) may also be **MPI\_ANY\_SOURCE**.

### Location

`./src/pt2pt/sendrecv.c`

### MPI\_Sendrecv\_replace

**MPI\_Sendrecv\_replace** — Sends and receives using a single buffer

### Synopsis

```
int MPI_Sendrecv_replace( void *buf, int count, MPI_Datatype datatype,
                           int dest, int sendtag, int source, int recvtag,
                           MPI_Comm comm, MPI_Status *status )
```

### Input Parameters

**count** number of elements in send and receive buffer (integer)  
**datatype** type of elements in send and receive buffer (handle)  
**dest** rank of destination (integer)  
**sendtag** send message tag (integer)  
**source** rank of source (integer)  
**recvtag** receive message tag (integer)  
**comm** communicator (handle)

### Output Parameters

**buf** initial address of send and receive buffer (choice)  
**status** status object (Status)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g. **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS** No error; MPI routine completed successfully.

**MPLERR\_COMM** Invalid communicator. A common error is to use a null communicator in a call

(not even allowed in **MPI\_Comm\_rank**).

**MPL\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPL\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPL\_ERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

**MPL\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

**MPL\_ERR\_TRUNCATE**

Message truncated on receive. The buffer size specified was too small for the received message. This is a recoverable error in the MPICH implementation.

**MPL\_ERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**Location**

`./src/pt2pt/sendrecv_rep.c`

**MPI\_Ssend**

**MPI\_Ssend** — Basic synchronous send

**Synopsis**

```
int MPI_Ssend( void *buf, int count, MPI_Datatype datatype,
               int dest, int tag, MPI_Comm comm )
```

**Input Parameters**

**buf** initial address of send buffer (choice)  
**count** number of elements in send buffer (nonnegative integer)  
**datatype** datatype of each send buffer element (handle)  
**dest** rank of destination (integer)  
**tag** message tag (integer)  
**comm** communicator (handle)

**Notes for Fortran**

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

**Errors**

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPL\_SUCCESS**

No error; MPI routine completed successfully.

**MPL\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPL\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPL\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPL\_ERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

**MPL\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

**Location**

`./src/pt2pt/ssend.c`

**MPI\_Ssend\_init****MPI\_Ssend\_init**

**MPI\_Ssend\_init** — Builds a handle for a synchronous send

**Synopsis**

```
int MPI_Ssend_init( void *buf, int count, MPI_Datatype datatype, int dest,
                    int tag, MPI_Comm comm, MPI_Request *request )
```

**Input Parameters**

**buf** initial address of send buffer (choice)  
**count** number of elements sent (integer)  
**datatype** type of each element (handle)  
**dest** rank of destination (integer)  
**tag** message tag (integer)  
**comm** communicator (handle)

Output Parameter

request            communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in `MPI_Comm_rank`).

**MPI\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see `MPI_Type_commit`).

**MPI\_ERR\_TAG**

Invalid tag argument. Tags must be non-negative; tags in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_TAG`. The largest tag value is available through the attribute `MPI_TAG_UB`.

**MPI\_ERR\_RANK**

Invalid source or destination rank. Ranks must be between zero and the size of the communicator minus one; ranks in a receive (`MPI_Recv`, `MPI_Irecv`, `MPI_Sendrecv`, etc.) may also be `MPI_ANY_SOURCE`.

Location

`./src/pt2pt/ssend_init.c`

**MPI\_Start**

**MPI\_Start** — Initiates a communication with a persistent request handle

Synopsis

```
int MPI_Start(  
    MPI_Request *request)
```

Input Parameter

request            communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_REQUEST**

Invalid MPI\_Request. Either null or, in the case of a `MPI_Start` or `MPI_Startall`, not a persistent request.

Location

`./src/pt2pt/start.c`

**MPI\_Startall**

**MPI\_Startall** — Starts a collection of requests

Synopsis

```
int MPI_Startall( int count, MPI_Request array_of_requests[] )
```

Input Parameters

count            list length (integer)

array\_of\_requests    array of requests (array of handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

./src/pt2pt/startall.c

MPI\_Status\_c2f

MPI\_Status\_c2f — Convert a C status to a Fortran status

Synopsis

int MPI\_Status\_c2f( MPI\_Status \*c\_status, MPI\_Fint \*f\_status )

Input Parameters

c\_status      Status value in C (Status)

Output Parameter

f\_status      Status value in Fortran (Integer)

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g. **MPI\_ERR\_RANK**).

Location

./src/misc2/statusc2f.c

MPI\_Status\_set\_cancelled

MPI\_Status\_set\_cancelled — Set the opaque part of an MPI\_Status so that MPI\_Test\_cancelled will return flag

Synopsis

int MPI\_Status\_set\_cancelled( MPI\_Status \*status, int flag )

Input Parameters

status      Status to associate count with (Status)  
flag      if true indicates that request was cancelled (logical)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_Wtime** and **MPI\_Wtick**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

Location

./src/external/statuscancel.c

MPI\_Status\_set\_elements

MPI\_Status\_set\_elements

MPI\_Status\_set\_elements — Set the opaque part of an MPI\_Status so that MPI\_Get\_elements will return count.

Synopsis

int MPI\_Status\_set\_elements( MPI\_Status \*status, MPI\_Datatype datatype, int count )

Input Parameters

status      Status to associate count with (Status)  
datatype    datatype associated with count (handle)  
count      number of elements to associate with status (integer)

Location

./src/external/statuselm.c

MPI_Test	MPI_Test
----------	----------

**MPI\_Test** — Tests for the completion of a send or receive

Synopsis

```
int MPI_Test (
    MPI_Request *request,
    int *flag,
    MPI_Status *status)
```

Input Parameter

**request**      communication request (handle)

Output Parameter

**flag**          true if operation completed (logical)  
**status**        status object (Status). May be **MPI\_STATUS\_IGNORE**.

Note on status for send operations

For send operations, the only use of status is for **MPI\_Test\_cancelled** or in the case that there is an error, in which case the **MPI\_ERROR** field of status will be set.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**      No error; MPI routine completed successfully.

**MPI\_ERR\_REQUEST**   Invalid **MPI\_Request**. Either null or, in the case of a **MPI\_Start** or **MPI\_Startall**,

not a persistent request.

**MPI\_ERR\_ARG**       Invalid argument. Some argument is invalid and is not identified by a specific

error class (e.g., **MPI\_ERR\_RANK**).

Location

`./src/pt2pt/test.c`

MPI\_Test\_cancelled

MPI\_Test\_cancelled

**MPI\_Test\_cancelled** — Tests to see if a request was cancelled

Synopsis

```
int MPI_Test_cancelled(
    MPI_Status *status,
    int *flag)
```

Input Parameter

**status**        status object (Status)

Output Parameter

**flag**          (logical)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./src/pt2pt/testcancel.c`

MPI\_Testall

MPI\_Testall

**MPI\_Testall** — Tests for the completion of all previously initiated communications

Synopsis

```
int MPI_Testall(
    int count,
    MPI_Request array_of_requests[],
    int *flag,
    MPI_Status array_of_statuses[] )
```

### Input Parameters

**count**                    lists length (integer)  
**array\_of\_requests**      array of requests (array of handles)

### Output Parameters

**flag**                    (logical)  
**array\_of\_statuses**      array of status objects (array of Status). May be **MPI\_STATUSES\_IGNORE**.

### Notes

**flag** is true only if all requests have completed. Otherwise, **flag** is false and neither the **array\_of\_requests** nor the **array\_of\_statuses** is modified.

### Note on status for send operations

For send operations, the only use of status is for **MPI\_Test\_cancelled** or in the case that there is an error, in which case the **MPI\_ERROR** field of status will be set.

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

#### **MPI\_SUCCESS**

No error; MPI routine completed successfully.

#### **MPI\_ERR\_IN\_STATUS**

The actual error value is in the **MPI\_Status** argument. This error class is returned only from the multiple-completion routines (**MPI\_Testall**, **MPI\_Testany**, **MPI\_Testsome**, **MPI\_Waitall**, **MPI\_Waitany**, and **MPI\_Waitsome**). The field **MPI\_ERROR** in the status argument contains the error value or **MPI\_SUCCESS** (no error and complete) or **MPI\_ERR\_PENDING** to indicate that the request has not completed.

The MPI Standard does not specify what the result of the multiple completion routines is when an error occurs. For example, in an **MPI\_WAITALL**, does the routine wait for all requests to either fail or complete, or does it return immediately (with the MPI definition of immediately, which means independent of actions of other MPI processes)? MPICH has chosen to make the return immediate (alternately, local in MPI terms), and to use the error class **MPI\_ERR\_PENDING** (introduced in MPI 1.1) to indicate which requests have not completed. In most cases, only one request with an error

will be detected in each call to an MPI routine that tests multiple requests. The requests that have not been processed (because an error occurred in one of the requests) will have their **MPI\_ERROR** field marked with **MPI\_ERR\_PENDING**.

### Location

`./src/pt2pt/testall.c`

<b>MPI_Testany</b>	<b>MPI_Testany</b>
--------------------	--------------------

**MPI\_Testany** — Tests for completion of any previously initiated communication

### Synopsis

```
int MPI_Testany(
    int count,
    MPI_Request array_of_requests[],
    int *index, int *flag,
    MPI_Status *status )
```

### Input Parameters

**count**                    list length (integer)  
**array\_of\_requests**      array of requests (array of handles)

### Output Parameters

**index**                    index of operation that completed, or **MPI\_UNDEFINED** if none completed (integer)  
**flag**                    true if one of the operations is complete (logical)  
**status**                  status object (Status). May be **MPI\_STATUS\_IGNORE**.

### Note on status for send operations

For send operations, the only use of status is for **MPI\_Test\_cancelled** or in the case that there is an error, in which case the **MPI\_ERROR** field of status will be set.

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

Location

`./src/pt2pt/testany.c`

MPI\_Testsome

MPI\_Testsome — Tests for some given communications to complete

Synopsis

```
int MPI_Testsome(
    int incount,
    MPI_Request array_of_requests[],
    int *outcount,
    int array_of_indices[],
    MPI_Status array_of_statuses[] )
```

Input Parameters

**incount**      length of array\_of\_requests (integer)  
**array\_of\_requests**      array of requests (array of handles)

Output Parameters

**outcount**      number of completed requests (integer)  
**array\_of\_indices**      array of indices of operations that completed (array of integers)  
**array\_of\_statuses**      array of status objects for operations that completed (array of Status). May be MPI\_STATUSES\_IGNORE.

Note on status for send operations

For send operations, the only use of status is for `MPI_Test_cancelled` or in the case that there is an error, in which case the `MPI_ERROR` field of status will be set.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_IN\_STATUS

The actual error value is in the `MPI_Status` argument. This error class is returned only from the multiple-completion routines (`MPI_Testall`, `MPI_Testany`, `MPI_Testsome`, `MPI_Waitall`, `MPI_Waitany`, and `MPI_Waitsome`). The field `MPI_ERROR` in the status argument contains the error value or `MPI_SUCCESS` (no error and complete) or `MPI_ERR_PENDING` to indicate that the request has not completed.

The MPI Standard does not specify what the result of the multiple completion routines is when an error occurs. For example, in an `MPI_WAITALL`, does the routine wait for all requests to either fail or complete, or does it return immediately (with the MPI definition of immediately, which means independent of actions of other MPI processes)? MPICH has chosen to make the return immediate (alternately, local in MPI terms) and to use the error class `MPI_ERR_PENDING` (introduced in MPI 1.1) to indicate which requests have not completed. In most cases, only one request with an error will be detected in each call to an MPI routine that tests multiple requests. The requests that have not been processed (because an error occurred in one of the requests) will have their `MPI_ERROR` field marked with `MPI_ERR_PENDING`.

Location

`./src/pt2pt/testsome.c`

MPI\_Topo\_test

MPI\_Topo\_test

MPI\_Topo\_test — Determines the type of topology (if any) associated with a communicator

Synopsis

```
int MPI_Topo_test ( MPI_Comm comm, int *top_type )
```

Input Parameter

**comm**      communicator (handle)

Output Parameter

top\_type      topology type of communicator comm (choice).

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**

Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

See Also

**MPI\_Graph\_create**, **MPI\_Cart\_create**

Location

`./src/topo1/topo_test.c`

MPI_Type_commit	MPI_Type_commit
<b>MPI_Type_commit</b> — Commits the datatype	

Synopsis

int **MPI\_Type\_commit** ( **MPI\_Datatype** \*datatype )

Input Parameter

datatype      datatype (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see **MPI\_Type\_commit**).

Location

`./src/pr2pt/type_commit.c`

MPI_Type_contiguous	MPI_Type_contiguous
<b>MPI_Type_contiguous</b> — Creates a contiguous datatype	

Synopsis

int **MPI\_Type\_contiguous**(  
    int count,  
    **MPI\_Datatype** old\_type,  
    **MPI\_Datatype** \*newtype)

Input Parameters

count          replication count (nonnegative integer)  
oldtype        old datatype (handle)

Output Parameter

newtype        new datatype (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return



value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPI\_ERR\_INTERR

This error is returned when some part of the MPICH implementation is unable to acquire memory.

Location

`./src/pt2pt/type_contig.c`

MPI\_Type\_create\_darray

MPI\_Type\_create\_darray

**MPI\_Type\_create\_darray** — Creates a datatype corresponding to a distributed, multidimensional array

Synopsis

```
int MPI_Type_create_darray(int size, int rank, int ndims,  
                           int *array_of_gsizes, int *array_of_distribs,  
                           int *array_of_dargs, int *array_of_psize,  
                           int order, MPI_Datatype oldtype,  
                           MPI_Datatype *newtype)
```

Input Parameters

**size**           size of process group (positive integer)  
**rank**          rank in process group (nonnegative integer)  
**ndims**        number of array dimensions as well as process grid dimensions (positive integer)  
**array\_of\_gsizes**   number of elements of type `oldtype` in each dimension of global array (array of positive integers)  
**array\_of\_distribs**   distribution of array in each dimension (array of state)

array\_of\_dargs

distribution argument in each dimension (array of positive integers)

array\_of\_psize

size of process grid in each dimension (array of positive integers)

order

array storage order flag (state)

oldtype

old datatype (handle)

Output Parameters

**newtype**       new datatype (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

`./src/misc2/darray.c`

MPI\_Type\_create\_indexed\_block

MPI\_Type\_create\_indexed\_block

**MPI\_Type\_create\_indexed\_block** — Creates an indexed datatype with constant sized blocks

Synopsis

```
int MPI_Type_create_indexed_block(  
    int count,  
    int blocklength,  
    int array_of_displacements[],  
    MPI_Datatype old_type,  
    MPI_Datatype *newtype )
```

Input Parameters

**count**        number of blocks – also number of entries in indices and blocklens  
**blocklength**   number of elements in each block (integer)  
**array\_of\_displacements**   displacement of each block in multiples of `old_type` (array of integer)  
**old\_type**     old datatype (handle)

Output Parameter

**newtype**       new datatype (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

The indices are displacements, and are based on a zero origin. A common error is to do something like to following

```
integer a(100)
integer blens(10), indices(10)
do i=1,10
  10  indices(i) = 1 + (i-1)*10
  call MPI_TYPE_CREATE_INDEXED_BLOCK(10,i,indices,MPI_INTEGER,newtype,ierr)
  call MPI_TYPE_COMMIT(newtype,ierr)
  call MPI_SEND(a,1,newtype,...)
```

expecting this to send `a(1),a(11),...` because the indices have values `1,11,...`. Because these are *displacements* from the beginning of `a`, it actually sends `a(1+1),a(1+11),...`. If you wish to consider the displacements as indices into a Fortran array, consider declaring the Fortran array with a zero origin

```
integer a(0:99)
```

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_ERR\_COUNT

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

MPI\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see

`MPI_Type_commit`).

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific

error class (e.g., `MPI_ERR_RANK`).

MPI\_ERR\_INTERR

This error is returned when some part of the MPICH implementation is unable to acquire memory.

Location

```
./src/misc2/type_bkind.c
```

MPI\_Type\_create\_subarray

MPI\_Type\_extent

MPI\_Type\_create\_subarray

`MPI_Type_create_subarray` — Creates a datatype describing a subarray of a multidimensional array

Synopsis

```
int MPI_Type_create_subarray(
  int ndims,
  int *array_of_sizes,
  int *array_of_subsizes,
  int *array_of_starts,
  int order,
  MPI_Datatype oldtype,
  MPI_Datatype *newtype)
```

Input Parameters

- `ndims` number of array dimensions (positive integer)
- `array_of_sizes` number of elements of type oldtype in each dimension of the full array (array of positive integers)
- `array_of_subsizes` number of elements of type oldtype in each dimension of the subarray (array of positive integers)
- `array_of_starts` starting coordinates of the subarray in each dimension (array of nonnegative integers)
- `order` array storage order flag (state)
- `oldtype` old datatype (handle)

Output Parameters

- `newtype` new datatype (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Location

```
./src/misc2/subarray.c
```

MPI\_Type\_extent

MPI\_Type\_extent

`MPI_Type_extent` — Returns the extent of a datatype

Synopsis

```
int MPI_Type_extent( MPI_Datatype datatype, MPI_Aint *extent )
```

Input Parameters

datatype      datatype (handle)

Output Parameter

extent          datatype extent (integer)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPL\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

Location

./src/pt2pt/type\_extent.c

MPI_Type_free	MPI_Type_free
MPI_Type_free — Frees the datatype	

Synopsis

int MPI\_Type\_free ( MPI\_Datatype \*datatype )

Input Parameter

datatype      datatype that is freed (handle)

Predefined types

The MPI standard states that (in Opaque Objects)

**MPI** provides certain predefined opaque objects and predefined, static handles to these objects. Such objects may not be destroyed.

Thus, it is an error to free a predefined datatype. The same section makes it clear that it is an error to free a null datatype.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr**  at the end of the argument list.  **ierr**  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPL\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

MPL\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g. **MPI\_ERR\_RANK**).

Location

./src/pt2pt/type\_free.c

MPI_Type_get_contents	MPI_Type_get_contents
MPI_Type_get_contents — Retrieves the actual arguments used in the creation call for a datatype	

Synopsis

```
int MPI_Type_get_contents(  
    MPI_Datatype datatype,  
    int max_integers,  
    int max_addresses,  
    int max_datatypes,  
    int *array_of_integers,  
    MPI_Aint *array_of_addresses,  
    MPI_Datatype *array_of_datatypes)
```

Input Parameters

- datatype** datatype to access (handle)
- max\_integers** number of elements in array\_of\_integers (non-negative integer)
- max\_addresses** number of elements in array\_of\_addresses (non-negative integer)
- max\_datatypes** number of elements in array\_of\_datatypes (non-negative integer)

Output Parameters

- array\_of\_integers** contains integer arguments used in constructing datatype (array of integers)
- array\_of\_addresses** contains address arguments used in constructing datatype (array of integers)
- array\_of\_datatypes** contains datatype arguments used in constructing datatype (array of handles)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./src/external/type\_get\_cont.c

MPI_Type_get_envelope	MPI_Type_get_envelope
<b>MPI_Type_get_envelope</b> — Returns information on the number and type of input arguments used in the call that created datatype	

Synopsis

```
int MPI_Type_get_envelope(  
    MPI_Datatype datatype,  
    int *num_integers,  
    int *num_addresses,  
    int *num_datatypes,  
    int *combiner)
```

Input Parameters

- datatype** datatype to access (handle)

Output Parameters

- num\_integers** number of input integers used in the call constructing combiner (nonnegative integer)
- num\_addresses** number of input addresses used in the call constructing combiner (nonnegative integer)
- num\_datatypes** number of input datatypes used in the call constructing combiner (nonnegative integer)
- combiner** combiner (state)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

./src/external/type\_get\_env.c

MPI_Type_hindexed	MPI_Type_hindexed
<b>MPI_Type_hindexed</b> — Creates an indexed datatype with offsets in bytes	

Synopsis

```
int MPI_Type_hindexed(  
    int count,  
    int blocklens[],  
    MPI_Aint indices[],  
    MPI_Datatype old_type,  
    MPI_Datatype *newtype )
```

Input Parameters

- count** number of blocks – also number of entries in indices and blocklens
- blocklens** number of elements in each block (array of nonnegative integers)
- indices** byte displacement of each block (array of MPI\_Aint)
- old\_type** old datatype (handle)

Output Parameter

- newtype** new datatype (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Also see the discussion for `MPI_Type_indexed` about the `indices` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPI\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_INTERN**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/pt2pt/type_hind.c`

<code>MPI_Type_hvector</code>	<code>MPI_Type_hvector</code>
<code>MPI_Type_hvector</code> — Creates a vector (strided) datatype with offset in bytes	
<b>Synopsis</b>	
<pre>int MPI_Type_hvector(     int count,     int blocklen,     MPI_Aint stride,     MPI_Datatype old_type,     MPI_Datatype *newtype )</pre>	

Input Parameters

`count`      number of blocks (nonnegative integer)

**blocklength**      number of elements in each block (nonnegative integer)  
**stride**            number of bytes between start of each block (integer)  
**old\_type**          old datatype (handle)

Output Parameter

**newtype**          new datatype (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_Wtime` and `MPI_Wtick`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPI\_ERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

**MPI\_ERR\_INTERN**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

Location

`./src/pt2pt/type_hvec.c`

<code>MPI_Type_indexed</code>	<code>MPI_Type_indexed</code>
<code>MPI_Type_indexed</code> — Creates an indexed datatype	
<b>Synopsis</b>	
<pre>int MPI_Type_indexed(     int count,     int blocklens[],     int indices[],</pre>	

**MPI\_Datatype old\_type,**  
**MPI\_Datatype \*newtype )**

### Input Parameters

**count**      number of blocks – also number of entries in indices and blocklens  
**blocklens**    number of elements in each block (array of nonnegative integers)  
**indices**      displacement of each block in multiples of **old\_type** (array of integers)  
**old\_type**    old datatype (handle)

### Output Parameter

**newtype**    new datatype (handle)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

The indices are displacements, and are based on a zero origin. A common error is to do something like to following

```
integer a(100)
integer blens(10), indices(10)
do i=1,10
  blens(i) = 1
  indices(i) = 1 + (i-1)*10
  call MPI_TYPE_INDEXED(10,blens,indices,MPI_INTEGER,newtype,ierr)
  call MPI_TYPE_COMMIT(newtype,ierr)
  call MPI_SEND(a,1,newtype,...)
```

expecting this to send **a(1),a(11),...** because the indices have values **1,11,...**. Because these are *displacements* from the beginning of **a**, it actually sends **a(1+1),a(1+11),...**.

If you wish to consider the displacements as indices into a Fortran array, consider declaring the Fortran array with a zero origin

```
integer a(0:99)
```

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPLERR\_COUNT**

No error; MPI routine completed successfully.  
 Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPLERR\_TYPE**

Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see **MPI\_Type\_commit**).

Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see **MPI\_Type\_commit**).

**MPLERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

**MPLERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

### Location

`./src/pt2pt/type_ind.c`

---

**MPI\_Type\_lb**

**MPI\_Type\_lb**

**MPI\_Type\_lb** — Returns the lower-bound of a datatype

### Synopsis

**int MPI\_Type\_lb ( MPI\_Datatype datatype, MPI\_Aint \*displacement )**

### Input Parameters

**datatype**    datatype (handle)

### Output Parameter

**displacement**    displacement of lower bound from origin, in bytes (integer)

### Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.

All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

### Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPLSUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_TYPE**

Invalid datatype argument. May be an uncommitted **MPI\_Datatype** (see **MPI\_Type\_commit**).

MPL\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/pt2pt/type_lb.c`

MPL\_Type\_size

**MPL\_Type\_size** — Return the number of bytes occupied by entries in the datatype

Synopsis

`int MPI_Type_size ( MPI_Datatype datatype, int *size )`

Input Parameters

`datatype`      datatype (handle)

Output Parameter

`size`            datatype size (integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr` at the end of the argument list.  `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

MPL\_SUCCESS

No error; MPI routine completed successfully.

MPL\_ERR\_TYPE

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

MPL\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/pt2pt/type_size.c`

MPL\_Type\_struct

**MPL\_Type\_struct** — Creates a struct datatype

Synopsis

```
int MPI_Type_struct(
    int count,
    int blocklens[],
    MPI_Aint indices[],
    MPI_Datatype old_types[],
    MPI_Datatype *newtype )
```

Input Parameters

`count`            number of blocks (integer) – also number of entries in arrays `array_of_types`, `array_of_displacements` and `array_of_blocklengths`  
`blocklens`        number of elements in each block (array)  
`indices`          byte displacement of each block (array)  
`old_types`        type of elements in each block (array of handles to datatype objects)

Output Parameter

`newtype`          new datatype (handle)

Notes

If an upperbound is set explicitly by using the MPI datatype `MPI_UB`, the corresponding index must be positive.

The MPI standard originally made vague statements about padding and alignment; this was intended to allow the simple definition of structures that could be sent with a count greater than one. For example,

```
struct { int a; char b; } foo;
```

may have `sizeof(foo) > sizeof(int) + sizeof(char)`; for example, `sizeof(foo) == 2*sizeof(int)`. The initial version of the MPI standard defined the extent of a datatype as including an *epsilon* that would have allowed an implementation to make the extent an MPI datatype for this structure equal to `2*sizeof(int)`. However, since different systems might define different paddings, there was much discussion by the MPI Forum about what was the correct value of epsilon, and one suggestion was to define epsilon as zero. This would have been the best thing to do in MPI 1.0, particularly since the `MPI_UB` type allows the user to easily set the end of the structure. Unfortunately, this change did not make it into the final document. Currently, this routine does not add any padding; since the amount of padding needed is determined by the compiler that the user is using to build their code, not the compiler used to construct the MPI library. A later version of MPICH may provide for some natural choices of padding (e.g., multiple of the size of the largest basic member), but users are advised to never depend on this, even with

vendor MPI implementations. Instead, if you define a structure datatype and wish to send or receive multiple items, you should explicitly include an `MPI_UB` entry as the last member of the structure. For example, the following code can be used for the structure `foo`

```
blen[0] = 1; indices[0] = 0; oldtypes[0] = MPI_INT;
blen[1] = 1; indices[1] = &foo.b - &foo; oldtypes[1] = MPI_CHAR;
blen[2] = 1; indices[2] = sizeof(foo); oldtypes[2] = MPI_UB;
MPI_Type_struct( 3, blen, indices, oldtypes, &newtype );
```

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPLSUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPLERR\_COUNT**

Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPLERR\_INTERR**

This error is returned when some part of the MPICH implementation is unable to acquire memory.

Location

```
./src/pr2pt/type_struct.c
```

MPI_Type_ub	MPI_Type_ub
-------------	-------------

**MPI\_Type\_ub** — Returns the upper bound of a datatype

Synopsis

```
int MPI_Type_ub ( MPI_Datatype datatype, MPI_Aint *displacement )
```

Input Parameters

`datatype`      datatype (handle)

Output Parameter

`displacement`   displacement of upper bound from origin, in bytes (integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument  `ierr`  at the end of the argument list.  `ierr`  is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the  `call`  statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPLSUCCESS**

No error; MPI routine completed successfully.

**MPLERR\_TYPE**

Invalid datatype argument. May be an uncommitted `MPI_Datatype` (see `MPI_Type_commit`).

**MPLERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

```
./src/pr2pt/type_ub.c
```

MPI_Type_vector	MPI_Type_vector
-----------------	-----------------

**MPI\_Type\_vector** — Creates a vector (strided) datatype

Synopsis

```
int MPI_Type_vector(
    int count,
    int blocklen,
    int stride,
    MPI_Datatype old_type,
    MPI_Datatype *newtype )
```



Input Parameters

**count**      number of blocks (nonnegative integer)  
**blocklength**      number of elements in each block (nonnegative integer)  
**stride**      number of elements between start of each block (integer)  
**oldtype**      old datatype (handle)

Output Parameter

**newtype**      new datatype (handle)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Location

`./src/pt2pt/type_vec.c`

MPI_Unpack	MPI_Unpack
MPI_Unpack — Unpack a datatype into contiguous memory	

Synopsis

```
int MPI_Unpack ( void *inbuf, int insize, int *position,  
                 void *outbuf, int outcount, MPI_Datatype datatype,  
                 MPI_Comm comm )
```

Input Parameters

**inbuf**      input buffer start (choice)  
**insize**      size of input buffer, in bytes (integer)  
**position**      current position in bytes (integer)  
**outcount**      number of items to be unpacked (integer)  
**datatype**      datatype of each output data item (handle)  
**comm**      communicator for packed message (handle)

Output Parameter

**outbuf**      output buffer start (choice)

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument **ierr** at the end of the argument list. **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**      No error; MPI routine completed successfully.

**MPI\_ERR\_COMM**      Invalid communicator. A common error is to use a null communicator in a call (not even allowed in **MPI\_Comm\_rank**).

**MPI\_ERR\_COUNT**      Invalid count argument. Count arguments must be non-negative; a count of zero is often valid.

**MPI\_ERR\_TYPE**      Invalid datatype argument. May be an uncommitted MPI\_Datatype (see **MPI\_Type\_commit**).

**MPI\_ERR\_ARG**      Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

See Also

**MPI\_Pack**, **MPI\_Pack\_size**

Location

`./src/pt2pt/unpack.c`

MPI_Wait	MPI_Wait
MPI_Wait — Waits for an MPI send or receive to complete	

Synopsis

```
int MPI_Wait ( MPI_Request *request,  
               MPI_Status *status)
```

Input Parameter

request            request (handle)

Output Parameter

status            status object (Status) . May be MPI\_STATUS\_IGNORE.

Note on status for send operations

For send operations, the only use of status is for MPI\_Test\_cancelled or in the case that there is an error, in which case the MPI\_ERROR field of status will be set.

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Errors

All MPI routines (except MPI\_Wtime and MPI\_Wtick) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with MPI\_Errhandler\_set; the predefined error handler MPI\_ERRORS\_RETURN may be used to cause error values to be returned. Note that MPI does not guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_REQUEST

Invalid MPI\_Request. Either null or, in the case of a MPI\_Start or MPI\_Startall, not a persistent request.

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., MPI\_ERR\_RANK).

Location

./src/pt2pt/wait.c

MPI\_Waitall

MPI\_Waitall

MPI\_Waitall — Waits for all given communications to complete

Synopsis

```
int MPI_Waitall(
    int count,
    MPI_Request array_of_requests[],
    MPI_Status array_of_statuses[] )
```

Input Parameters

count            lists length (integer)  
array\_of\_requests    array of requests (array of handles)

Output Parameter

array\_of\_statuses    array of status objects (array of Status). May be MPI\_STATUSES\_IGNORE

Note on status for send operations

For send operations, the only use of status is for MPI\_Test\_cancelled or in the case that there is an error, in which case the MPI\_ERROR field of status will be set.

Notes for Fortran

All MPI routines in Fortran (except for MPI\_WTIME and MPI\_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.

All MPI objects (e.g., MPI\_Datatype, MPI\_Comm) are of type INTEGER in Fortran.

Errors

All MPI routines (except MPI\_Wtime and MPI\_Wtick) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with MPI\_Errhandler\_set; the predefined error handler MPI\_ERRORS\_RETURN may be used to cause error values to be returned. Note that MPI does not guarantee that an MPI program can continue past an error.

MPI\_SUCCESS

No error; MPI routine completed successfully.

MPI\_ERR\_REQUEST

Invalid MPI\_Request. Either null or, in the case of a MPI\_Start or MPI\_Startall, not a persistent request.

MPI\_ERR\_ARG

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., MPI\_ERR\_RANK).

MPI\_ERR\_IN\_STATUS

The actual error value is in the MPI\_Status argument. This error class is returned only from the multiple-completion routines (MPI\_Testall, MPI\_Testany, MPI\_Testsome, MPI\_Waitall, MPI\_Waitany, and MPI\_Waitsome). The field MPI\_ERROR in the status argument contains the error value or MPI\_SUCCESS (no error and complete) or MPI\_ERR\_PENDING to indicate that the request has not completed.

The MPI Standard does not specify what the result of the multiple completion routines is when an error occurs. For example, in an MPI\_WAITALL, does the routine wait for all requests to either fail or complete, or does it return immediately (with the MPI definition of immediately, which means

independent of actions of other MPI processes)? `MPICH` has chosen to make the return immediate (alternately, local in MPI terms), and to use the error class `MPI_ERR_PENDING` (introduced in MPI 1.1) to indicate which requests have not completed. In most cases, only one request with an error will be detected in each call to an MPI routine that tests multiple requests. The requests that have not been processed (because an error occurred in one of the requests) will have their `MPI_ERROR` field marked with `MPI_ERR_PENDING`.

**MPI\_ERR\_PENDING**  
Pending request (not an error). See `MPI_ERR_IN_STATUS`. This value indicates that the request is not complete nor has a encountered a detected error.

Location

`./src/pr2pt/waitall.c`

MPI_Waitany	MPI_Waitany
-------------	-------------

**MPI\_Waitany** — Waits for any specified send or receive to complete

Synopsis

```
int MPI_Waitany(  
    int count,  
    MPI_Request array_of_requests[],  
    int *index,  
    MPI_Status *status )
```

Input Parameters

**count** list length (integer)  
**array\_of\_requests** array of requests (array of handles)

Output Parameters

**index** index of handle for operation that completed (integer). In the range 0 to `count-1`.  
In Fortran, the range is 1 to `count`.  
**status** status object (Status). May be `MPI_STATUS_IGNORE`.

Notes

If all of the requests are `MPI_REQUEST_NULL`, then `index` is returned as `MPI_UNDEFINED`, and `status` is returned as an empty status.

Note on status for send operations

For send operations, the only use of status is for `MPI_Test_cancelled` or in the case that there is an error, in which case the `MPI_ERROR` field of status will be set.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**

No error; MPI routine completed successfully.

**MPI\_ERR\_REQUEST**

Invalid `MPI_Request`. Either null or, in the case of a `MPI_Start` or `MPI_Startall`, not a persistent request.

**MPI\_ERR\_ARG**

Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., `MPI_ERR_RANK`).

Location

`./src/pr2pt/waitany.c`

MPI_Waitsome	MPI_Waitsome
--------------	--------------

**MPI\_Waitsome** — Waits for some given communications to complete

Synopsis

```
int MPI_Waitsome(  
    int incount,  
    MPI_Request array_of_requests[],  
    int *outcount,  
    int array_of_indices[],  
    MPI_Status array_of_statuses[] )
```

Input Parameters

**incount** length of `array_of_requests` (integer)  
**array\_of\_requests** array of requests (array of handles)

Output Parameters

- outcount**     number of completed requests (integer)
- array\_of\_indices**     array of indices of operations that completed (array of integers)
- array\_of\_statuses**     array of status objects for operations that completed (array of Status). May be **MPI\_STATUSES\_IGNORE**.

Notes

The array of indices are in the range 0 to **incount - 1** for C and in the range 1 to **incount** for Fortran.  
Null requests are ignored; if all requests are null, then the routine returns with **outcount** set to **MPI\_UNDEFINED**.

Note on status for send operations

For send operations, the only use of status is for **MPI\_Test**; cancelled or in the case that there is an error, in which case the **MPI\_ERROR** field of status will be set.

Notes for Fortran

All MPI routines in Fortran (except for **MPI\_WTIME** and **MPI\_WTICK**) have an additional argument  **ierr** at the end of the argument list.  **ierr** is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the **call** statement.  
All MPI objects (e.g., **MPI\_Datatype**, **MPI\_Comm**) are of type **INTEGER** in Fortran.

Errors

All MPI routines (except **MPI\_Wtime** and **MPI\_Wtick**) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with **MPI\_Errhandler\_set**; the predefined error handler **MPI\_ERRORS\_RETURN** may be used to cause error values to be returned. Note that MPI does *not* guarantee that an MPI program can continue past an error.

**MPI\_SUCCESS**     No error; MPI routine completed successfully.

**MPI\_ERR\_REQUEST**     Invalid **MPI\_Request**. Either null or, in the case of a **MPI\_Start** or **MPI\_Startall**,

**MPI\_ERR\_ARG**     not a persistent request.

**MPI\_ERR\_IN\_STATUS**     Invalid argument. Some argument is invalid and is not identified by a specific error class (e.g., **MPI\_ERR\_RANK**).

The actual error value is in the **MPI\_Status** argument. This error class is returned only from the multiple-completion routines (**MPI\_Testall**, **MPI\_Testany**, **MPI\_Testsome**, **MPI\_Waitall**, **MPI\_Waitany**, and **MPI\_Waitsome**). The field **MPI\_ERROR** in the status argument contains the error value or **MPI\_SUCCESS** (no error and complete) or **MPI\_ERR\_PENDING** to indicate that the request has not completed.

The MPI Standard does not specify what the result of the multiple completion routines is when an error occurs. For example, in an **MPI\_WAITALL**, does the routine wait for all requests to either fail or

complete, or does it return immediately (with the MPI definition of immediately, which means independent of actions of other MPI processes)? **MPICH** has chosen to make the return immediate (alternately, local in MPI terms), and to use the error class **MPI\_ERR\_PENDING** (introduced in MPI 1.1) to indicate which requests have not completed. In most cases, only one request with an error will be detected in each call to an MPI routine that tests multiple requests. The requests that have not been processed (because an error occurred in one of the requests) will have their **MPI\_ERROR** field marked with **MPI\_ERR\_PENDING**.

Location

`./src/pt2pt/waitsome.c`

**MPI\_Wtick**

**MPI\_Wtick** — Returns the resolution of **MPI\_Wtime**

Synopsis

`double MPI_Wtick()`

Return value

Time in seconds of resolution of **MPI\_Wtime**

Notes for Fortran

This is a function, declared as **DOUBLE PRECISION MPI\_WTICK()** in Fortran.

Location

`./src/env/wtick.c`

**MPI\_Wtime**

**MPI\_Wtime** — Returns an elapsed time on the calling processor

Synopsis

`double MPI_Wtime()`

Return value

Time in seconds since an arbitrary time in the past.

Notes

This is intended to be a high-resolution, elapsed (or wall) clock. See `MPI_WTICK` to determine the resolution of `MPI_WTIME`. If the attribute `MPI_WTIME_IS_GLOBAL` is defined and true, then the value is synchronized across all processes in `MPI_COMM_WORLD`.

Notes for Fortran

This is a function, declared as `DOUBLE PRECISION MPI_WTIME()` in Fortran.

See Also

also: `MPI_Wtick`, `MPI_Attr_get`

Location

`./src/env/wtime.c`

3 MPE routines

CLOG_Finalize	CLOG_Finalize
---------------	---------------

CLOG\_Finalize — Finalize CLOG logging

Synopsis

`void CLOG_Finalize( void )`

Location

`./mpe/src/clog.c`

CLOG_Init	CLOG_Init
-----------	-----------

CLOG\_Init — Initialize for CLOG logging

Synopsis

`void CLOG_Init( void )`

Location

`./mpe/src/clog.c`

CLOG_Output	CLOG_Output
-------------	-------------

CLOG\_Output — output a block of the log. The byte ordering, if needed, will be performed in this function using the conversion routines got from `Petsc`.

Synopsis

`void CLOG_output( buf )`  
`double *buf;`

Location

`./mpe/src/clog_merge.c`

CLOG_commtype	CLOG_commtype
---------------	---------------

CLOG\_commtype — print communicator creation event type

Synopsis

`void CLOG_commtype( etype )`  
`int etype;`

etype      event type for communicator creation event

Location

`./mpe/src/clog_util.c`

CLOG_cput	CLOG_cput
-----------	-----------

CLOG\_cput — move a log record from one of the input buffers to the output

Synopsis

`void CLOG_cput( ptr )`  
`double **ptr;`

This function moves records from one of the three buffers being merged into the output buffer. When the output buffer is filled, it is sent to the parent. A separate output routine handles output on the root. If the input buffer is emptied (endblock record is read) and corresponds to a child, a new buffer is received from the child. When an endlog record is read on the input buffer, the number

of sources is decremented and the time is set to positive infinity so that the empty input source will never have the lowest time.  
At entry we assume that \*p is pointing to a log record that is not an end-of-block record, and that outbuf is pointing to a buffer that has room in it for the record. We ensure that these conditions are met on exit as well, by sending (or writing, if we are the root) and receiving blocks as necessary.  
Input parameters  
**pointer to the record to be moved into the output buffer**

**Location**

./mpe/src/clog\_merge.c

CLOG_csync	CLOG_csync
------------	------------

CLOG\_csync — synchronize clocks for adjusting times in merge

**Synopsis**

```
void CLOG_csync( root, diffs )
int root;
double diffs[];
```

This version is sequential and non-scalable. The root process serially synchronizes with each slave, using the first algorithm in Gropp, "Scalable clock synchronization on distributed processors without a common clock". The array is calculated on the root but broadcast and returned on all processes.

**Inout Parameters**

**root**            process to serve as master  
**timediffs**     array of doubles to be filled in

**Location**

./mpe/src/clog\_merge.c

CLOG_get_new_event	CLOG_get_new_event
--------------------	--------------------

CLOG\_get\_new\_event — obtain unused event id

**Synopsis**

```
int CLOG_get_new_event( void )
```

**Location**

./mpe/src/clog.c

CLOG_get_new_state	CLOG_get_new_state
--------------------	--------------------

CLOG\_get\_new\_state — obtain unused state id

**Synopsis**

```
int CLOG_get_new_state( void )
```

**Location**

./mpe/src/clog.c

CLOG_init_buffers	CLOG_init_buffers
-------------------	-------------------

CLOG\_init\_buffers — initialize necessary buffers for clog logging.

**Synopsis**

```
void CLOG_init_buffers( void )
```

**Location**

./mpe/src/clog.c

CLOG_mergelogs	CLOG_mergelogs
----------------	----------------

CLOG\_mergelogs — merge individual logfiles into one via messages

**Synopsis**

```
void CLOG_mergelogs( shift, exccfilename, logtype )
int shift;
char *exccfilename;
int logtype;
```

first argument says whether to do time-shifting or not second arg is filename  
On process 0 in MPI\_COMM\_WORLD, collect logs from other processes and merge them with own log. Timestamps are assumed to be already adjusted on both incoming logs and the master's. On the other processes, fill in length and process id's and send them, a block at a time, to the master. The master writes out the merged log.

**Location**

./mpe/src/clog\_merge.c

CLOG_mergend	CLOG_mergend
CLOG_mergend — finish log processing	

Synopsis

void CLOG\_mergend()

Location

./mpe/src/clog\_merge.c

CLOG_msgtype	CLOG_msgtype
CLOG_msgtype — print communication event type	

Synopsis

void CLOG\_msgtype( etype )  
int etype;

etype            event type for pt2pt communication event

Location

./mpe/src/clog\_util.c

CLOG_newbuff	CLOG_newbuff
CLOG_newbuff — get and initialize new block of buffer	

Synopsis

void CLOG\_newbuff( CLOG\_BLOCK \*\*bufptr )

Input Parameter

bufptr           pointer to be filled in with address of new block

Location

./mpe/src/clog.c

CLOG_nodebuffer2disk	CLOG_nodebuffer2disk
CLOG_nodebuffer2disk — dump buffers into temporary log file.	

Synopsis

void CLOG\_nodebuffer2disk( void )

Location

./mpe/src/clog.c

CLOG_procbuf	CLOG_procbuf
CLOG_procbuf — postprocess a buffer of log records before merging	

Synopsis

void CLOG\_procbuf( buf )  
double \*buf;

This function fills in fields in log records that were left out during actual logging to save memory accesses. Typical fields are the process id and the lengths of records that are known by predefined type. This is also where we will adjust timestamps.

Input parameter

address of the buffer to be processed

Location

./mpe/src/clog\_merge.c

CLOG_reclen	CLOG_reclen
CLOG_reclen — get length (in doubles) of log record by type	

Synopsis

int CLOG\_reclen( type )  
int type;

Location

./mpe/src/clog\_util.c

CLOG_rectype	CLOG_rectype
CLOG_rectype — print log record type	

Synopsis

void CLOG\_rectype( type )  
int type;

rtype                    record type

Location

./mpe/src/clog\_util.c

CLOG_reinit_buff	CLOG_reinit_buff
------------------	------------------

CLOG\_reinit\_buff — reads CLOG\_BLOCKS from temporary logfile into memory.

Synopsis

void CLOG\_reinit\_buff( )

Location

./mpe/src/clog\_merge.c

CLOG_treestup	CLOG_treestup
---------------	---------------

CLOG\_treestup — locally determine parent and children in binary tree

Synopsis

void CLOG\_treestup( self, numprocs, myparent, mylchild, myrchild )  
int self, numprocs, \*myparent, \*mylchild, \*myrchild;

Input parameters

- self                    calling process's id
- np                      total number of processes in tree
- Output parameters
- parent                  parent in binary tree (or -1 if root)
- lchild                   left child in binary tree (or -1 if none)
- rchild                   right child in binary tree (or -1 if none)

Location

./mpe/src/clog\_merge.c

MPE_Add_RGB_color	MPE_Add_RGB_color
-------------------	-------------------

MPE\_Add\_RGB\_color — Adds a color to the colormap given its RGB values

Synopsis

```
#include "mpe.h"
int MPE_Add_RGB_color( graph, red, green, blue, mapping )
MPE_XGraph graph;
int red, green, blue;
MPE_Color *mapping;
```

Input Parameters

graph                   MPE graphics handle  
red, green, blue        color levels from 0 to 65535

Output Parameter

mapping                index of the new color

Return Values

-1                      maxcolors too large (equal to numcolors)

MPE\_SUCCESS           successful

mapping                index of the new color

Notes

This call adds a color cell to X11's color table, increments maxcolors (the index), and writes it to the mapping parameter.

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.

All MPI and MPE objects, MPI\_Comm, MPE\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.



Location

./mpe/src/mpe\_graphics.c

MPE_CaptureFile	MPE_CaptureFile
MPE_CaptureFile—Sets the base filename used to capture output from updates	

Synopsis

```
#include "mpe.h"
int MPE_CaptureFile( handle, fname, freq )
MPE_XGraph handle;
char *fname;
int freq;
```

Input Parameters

handle MPE graphics handle  
fname base file name (see below)  
freq Frequency of updates

Return Values

MPE\_ERR\_LOW\_MEM malloc for copy of the filename (fname) failed  
MPE\_ERR\_BAD\_ARGS handle parameter is bad  
MPE\_SUCCESS success

Notes

The output is written in xwd format to **fname**%d, where %d is the number of the file (starting from zero).

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Additional Notes for Fortran Interface

The trailing blanks in Fortran CHARACTER string argument will be ignored.

Location

./mpe/src/mpe\_graphics.c

MPE_Close_graphics	MPE_Close_graphics
MPE_Close_graphics — Closes an X11 graphics device	

Synopsis

```
#include "mpe.h"
int MPE_Close_graphics( handle )
MPE_XGraph *handle;
```

Input Parameter

handle MPE graphics handle.

Return Values

MPE\_ERR\_BAD\_ARGS handle parameter is bad  
MPE\_SUCCESS success

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE_Comm_global_rank	MPE_Comm_global_rank
MPE_Comm_global_rank — Returns the rank in MPI_COMM_WORLD for a given (communicator,rank) pair	

Synopsis

```
void MPE_Comm_global_rank( comm, rank, grank )
MPI_Comm comm;
int rank, *grank;
```

Input Parameters

comm           Communicator  
rank           Rank in comm

Output Parameters

grank           Rank in comm world

Location

./mpe/src/getgrank.c

---

MPE\_Counter\_create

---

MPE\_Counter\_create — create and initialize shared counter (process)

Synopsis

```
int MPE_Counter_create( oldcomm, smaller_comm, counter_comm )
MPI_Comm oldcomm, *smaller_comm, *counter_comm;
```

Input Parameter

oldcomm           Communicator to

Output Parameters

smaller\_comm

counter\_comm

Duplicate of oldcomm

Location

./mpe/src/mpe\_counter.c

---

MPE\_Counter\_free

---

MPE\_Counter\_free — free communicators associated with counter

Synopsis

```
int MPE_Counter_free( smaller_comm, counter_comm )
MPI_Comm *smaller_comm;
MPI_Comm *counter_comm;
```

Location

./mpe/src/mpe\_counter.c

---

MPE\_Counter\_nxtval

---

MPE\_Counter\_nxtval — obtain next value from shared counter, and update

Synopsis

```
int MPE_Counter_nxtval(counter_comm, value)
MPI_Comm counter_comm;
int *value;
```

Location

./mpe/src/mpe\_counter.c

---

MPE\_DecompId

---

MPE\_DecompId — Compute a balanced decomposition of a 1-D array

Synopsis

```
int MPE_DecompId( n, size, rank, s, e )
int n, size, rank, *s, *e;
```

Input Parameters

n               Length of the array  
size            Number of processors in decomposition  
rank            Rank of this processor in the decomposition (0 <= rank < size)

Output Parameters

s,e             Array indices are s:e, with the original array considered as 1:n.

Location

./mpe/src/decomp.c

---

MPE\_Describe\_event

---

MPE\_Describe\_event — Create log record describing an event type

Synopsis

```
int MPE_Describe_event( event, name )
int event;
char *name;
```

Input Parameters

event            Event number  
name            String describing the event.

See Also

MPE\_Log\_get\_event\_number

Location

./mpe/src/mpe\_log.c

MPE_Describe_state	MPE_Describe_state
MPE_Describe_state — Create log record describing a state	

Synopsis

```
int MPE_Describe_state( start, end, name, color )
int start, end;
char *name, *color;
```

Input Parameters

start           event number for the start of the state  
end            event number for the end of the state  
name           Name of the state  
color          color to display the state in

Notes

Adds string containing a state def to the logfile. The format of the definition is (in ALOG)

(LOG\_STATE\_DEF) 0 sevent; eevent 0 0 "color" "name"

States are added to a log file by calling MPE\_Log\_event for the start and end event numbers.

See Also

MPE\_Log\_get\_event\_number

Location

./mpe/src/mpe\_log.c

MPE_Draw_circle	MPE_Draw_circle
MPE_Draw_circle — Draws a circle	

Synopsis

```
#include "mpe.h"
int MPE_Draw_circle( graph, centerx, centery, radius, color )
MPE_XGraph graph;
int centerx, centery, radius;
MPE_Color color;
```

Input Parameters

graph           MPE graphics handle  
centerx        horizontal center point of the circle  
centery        vertical center point of the circle  
radius         radius of the circle  
color          color of the circle

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE_Draw_line	MPE_Draw_line
MPE_Draw_line — Draws a line on an X11 display	

Synopsis

```
#include "mpe.h"
int MPE_Draw_line( handle, x1, y_1, x2, y_2, color )
MPE_XGraph handle;
int        x1, y_1, x2, y_2;
MPE_Color color;
```

Input Parameters

**handle** MPE graphics handle  
**x1,y\_1** pixel position of one end of the line to draw. Coordinates are upper-left origin (standard X11)  
**x2,y\_2** pixel position of the other end of the line to draw. Coordinates are upper-left origin (standard X11)  
**color** Color *index* value. See **MPE\_MakeColorArray**. By default, the colors **MPE\_WHITE**, **MPE\_BLACK**, **MPE\_RED**, **MPE\_YELLOW**, **MPE\_GREEN**, **MPE\_CYAN**, **MPE\_BLUE**, **MPE\_MAGENTA**, **MPE\_AQUAMARINE**, **MPE\_FORESTGREEN**, **MPE\_ORANGE**, **MPE\_VIOLET**, **MPE\_BROWN**, **MPE\_PINK**, **MPE\_CORAL** and **MPE\_GRAY** are defined.

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, *ierr*, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPI\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE\_Draw\_logic

MPE_Draw_logic
----------------

**MPE\_Draw\_logic** — Sets logical operation for laying down new pixels

Synopsis

```
#include "mpe.h"
int MPE_Draw_logic( graph, function )
MPE_XGraph graph;
int function;
```

Input Parameters

**graph** MPE graphics handle  
**function** integer specifying one of the following:  
**MPE\_LOGIC\_COPY** - no logic, just copy the pixel  
**MPE\_LOGIC\_XOR** - xor the new pixel with the existing one and many more... see **mpe\_graphics.h**

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, *ierr*, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPI\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE\_Draw\_point

MPE_Draw_point
----------------

**MPE\_Draw\_point** — Draws a point on an X Windows display

Synopsis

```
#include "mpe.h"
int MPE_Draw_point( handle, x, y, color )
MPE_XGraph handle;
int x, y;
MPE_Color color;
```

Input Parameters

**handle** MPE graphics handle  
**x,y** pixel position to draw. Coordinates are upper-left origin (standard X11)  
**color** Color *index* value. See **MPE\_MakeColorArray**. By default, the colors **MPE\_WHITE**, **MPE\_BLACK**, **MPE\_RED**, **MPE\_YELLOW**, **MPE\_GREEN**, **MPE\_CYAN**, **MPE\_BLUE**, **MPE\_MAGENTA**, **MPE\_AQUAMARINE**, **MPE\_FORESTGREEN**, **MPE\_ORANGE**, **MPE\_VIOLET**, **MPE\_BROWN**, **MPE\_PINK**, **MPE\_CORAL** and **MPE\_GRAY** are defined.

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, *ierr*, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPI\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE\_Draw\_points

MPE_Draw_points
-----------------

**MPE\_Draw\_points** — Draws points on an X Windows display

Synopsis

```
#include "mpe.h"
int MPE_Draw_points( handle, points, npoints )
MPE_XGraph handle;
MPE_Point *points;
int npoints;
```

Input Parameters

**handle** MPE graphics handle  
**points** list of points to draw  
**npoints** number of points to draw

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE_Draw_string	MPE_Draw_string
MPE_Draw_string — Draw a text string	

Synopsis

```
#include "mpe.h"
int MPE_Draw_string( graph, x, y, color, string )
MPE_XGraph graph;
int x, y;
MPE_Color color;
char *string;
```

Input Parameters

**graph** MPE graphics handle  
**x** x-coordinate of the origin of the string  
**y** y-coordinate of the origin of the string  
**color** color of the text  
**string** text string to be drawn

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Additional Notes for Fortran Interface

The trailing blanks in Fortran CHARACTER string argument will be ignored.

Location

./mpe/src/mpe\_graphics.c

MPE_Fill_circle	MPE_Fill_circle
MPE_Fill_circle — Fills a circle	

Synopsis

```
#include "mpe.h"
int MPE_Fill_circle( graph, centerx, centery, radius, color )
MPE_XGraph graph;
int centerx, centery, radius;
MPE_Color color;
```

Input Parameters

**graph** MPE graphics handle  
**centerx** horizontal center point of the circle  
**centery** vertical center point of the circle  
**radius** radius of the circle  
**color** color of the circle

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE_Fill_rectangle	MPE_Fill_rectangle
MPE_Fill_rectangle — Draws a filled rectangle on an X11 display	

Synopsis

```
#include "mpe.h"
int MPE_Fill_rectangle( handle, x, y, w, h, color )
MPE_XGraph handle;
int x, y, w, h;
MPE_Color color;
```

Input Parameters

**handle** MPE graphics handle  
**x,y** pixel position of the upper left (low coordinate) corner of the rectangle to draw.  
**w,h** width and height of the rectangle  
**color** Color *index* value. See `MPE_MakeColorArray`. By default, the colors `MPE_WHITE`, `MPE_BLACK`, `MPE_RED`, `MPE_YELLOW`, `MPE_GREEN`, `MPE_CYAN`, `MPE_BLUE`, `MPE_MAGENTA`, `MPE_AQUAMARINE`, `MPE_FORESTGREEN`, `MPE_ORANGE`, `MPE_VIOLET`, `MPE_BROWN`, `MPE_PINK`, `MPE_CORAL` and `MPE_GRAY` are defined.

Notes

This uses the X11 definition of width and height, so you may want to add 1 to both of them.

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument,  `ierr`, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the `CALL` statement.  
All MPI and MPE objects, `MPI_Comm`, `MPE_XGraph` and `MPE_Color`, are of type `INTEGER` in Fortran.

Location

`./mpe/src/mpe_graphics.c`

MPE_Finish_log	MPE_Finish_log
MPE_Finish_log — Send log to master, who writes it out	

Synopsis

```
int MPE_Finish_log( filename )
char *filename;
```

Notes

This routine dumps a logfile in `alog` or `clog` format. It is collective over `MPI_COMM_WORLD`. The default is `alog` format. To generate `clog` output, set the environment variable `MPE_LOG_FORMAT` to `CLOG`.

Location

`./mpe/src/mpe_log.c`

MPE_GetTags	MPE_GetTags
MPE_GetTags — Returns tags that can be used in communication with a communicator	

Synopsis

```
int MPE_GetTags( comm_in, ntags, comm_out, first_tag )
MPI_Comm comm_in, *comm_out;
int ntags, *first_tag;
```

Input Parameters

**comm\_in** Input communicator  
**ntags** Number of tags

Output Parameters

**comm\_out** Output communicator. May be `comm_in`.  
**first\_tag** First tag available

Returns

`MPI_SUCCESS` on success, `MPI` error class on failure.

Notes

This routine returns the requested number of tags, with the tags being `first_tag`, `first_tag+1`, ..., `first_tag+ntags-1`.  
These tags are guaranteed to be unique within `comm_out`.

See Also

`MPE_ReturnTags`

Location

`./mpe/src/privtags.c`

MPE_Get_mouse_press	MPE_Get_mouse_press
MPE_Get_mouse_press — Waits for mouse button press	

Synopsis

```
#include "mpe.h"
int MPE_Get_mouse_press( graph, x, y, button )
MPE_XGraph graph;
int *x, *y, *button;
```

Input Parameter

**graph** MPE graphics handle

Output Parameters

**x** horizontal coordinate of the point where the mouse button was pressed  
**y** vertical coordinate of the point where the mouse button was pressed  
**button** which button was pressed: **MPE\_BUTTON[1-5]**

Notes

This routine waits for mouse button press, blocking until the mouse button is pressed inside this MPE window. When pressed, returns the coordinate relative to the upper right of this MPE window and the button that was pressed.

Location

`./mpe/src/xmouse.c`

MPE_IO_Stdout_to_file	
MPE_IO_Stdout_to_file — Re-direct stdout to a file	

Synopsis

`void MPE_IO_Stdout_to_file( char *name, int mode )`

Parameters

**name** Name of file. If it contains `%d`, this value will be replaced with the rank of the process in **MPI\_COMM\_WORLD**.  
**mode** Mode to open the file in (see the man page for `open`). A common value is `0644` (Read/Write for owner, Read for everyone else). Note that this value is *anded* with your current `umask` value.

Notes

Some systems may complain when standard output (`stdout`) is closed.

Location

`./mpe/src/mpe_io.c`

MPE_Iget_mouse_press	
MPE_Iget_mouse_press — Checks for mouse button press	

Synopsis

```
#include "mpe.h"
int MPE_Iget_mouse_press( graph, x, y, button, wasPressed )
MPE_XGraph graph;
int *x, *y, *button, *wasPressed;
```

Input Parameter

**graph** MPE graphics handle

Output Parameters

**x** horizontal coordinate of the point where the mouse button was pressed  
**y** vertical coordinate of the point where the mouse button was pressed  
**button** which button was pressed: **MPE\_BUTTON[1-5]**  
**wasPressed** 1 if the button was pressed, 0 if not

Notes

Checks if the mouse button has been pressed inside this MPE window. If pressed, returns the coordinate relative to the upper right of this MPE window and the button that was pressed.

Location

`./mpe/src/xmouse.c`

MPE_Init_log	
MPE_Init_log — Initialize for logging	

Synopsis

`int MPE_Init_log()`

Notes

Initializes the MPE logging package. This must be called before any of the other MPE logging routines. It is collective over **MPI\_COMM\_WORLD**

See Also

`MPE_Finish_log`

Location

`./mpe/src/mpe_log.c`

MPE_Initialized_logging	
MPE_Initialized_logging — Indicate whether MPE_Init_log or MPE_Finish_log have been called.	

Synopsis

```
int MPE_Initialized_logging ( )
```

Returns

0 if MPE\_Init\_log has not been called, 1 if MPE\_Init\_log has been called but MPE\_Finish\_log has not been called, and 2 otherwise.

Location

```
./mpe/src/mpe_log.c
```

MPE_Line_thickness	MPE_Line_thickness
MPE_Line_thickness — Sets thickness of lines	

Synopsis

```
#include "mpe.h"
int MPE_Line_thickness( graph, thickness )
MPE_XGraph graph;
int thickness;
```

Input Parameters

**graph**        MPE graphics handle  
**thickness**   integer specifying how many pixels wide lines should be

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

```
./mpe/src/mpe_graphics.c
```

MPE_Log_event	MPE_Log_event
MPE_Log_event — Logs an event	

Synopsis

```
int MPE_Log_event(event, data, string)
int event, data;
char *string;
```

Input Parameters

**event**        Event number  
**data**        Integer data value  
**string**       Optional string describing event

Location

```
./mpe/src/mpe_log.c
```

MPE_Log_get_event_number	MPE_Log_get_event_number
MPE_Log_get_event_number — Gets an unused event number	

Synopsis

```
int MPE_Log_get_event_number( )
```

Returns

A value that can be provided to MPE\_Describe\_event or MPE\_Describe\_state which will define an event or state not used before.

Notes

This routine is provided to allow packages to ensure that they are using unique event numbers. It relies on all packages using this routine.

Location

```
./mpe/src/mpe_log.c
```

MPE_Log_receive	MPE_Log_receive
MPE_Log_receive — log the sending of a message	



Synopsis

```
int MPE_Log_receive( otherParty, tag, size )
int otherParty, tag, size;
```

Location

./mpe/src/mpe\_log.c

MPE_Log_send	MPE_Log_send
MPE_Log_send — Logs the sending of a message	

Synopsis

```
int MPE_Log_send( otherParty, tag, size )
int otherParty, tag, size;
```

Location

./mpe/src/mpe\_log.c

MPE_Make_color_array	MPE_Make_color_array
MPE_Make_color_array — Makes an array of color indices	

Synopsis

```
#include "mpe.h"
int MPE_Make_color_array( handle, ncolors, array )
MPE_XGraph handle;
int ncolors;
MPE_Color array[];
```

Input Parameters

handle MPE graphics handle  
nc Number of colors

Output Parameter

array Array of color indices

Notes

The new colors for a uniform distribution in hue space and replace the existing colors *except* for MPE\_WHITE and MPE\_BLACK.

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE_Num_colors	MPE_Num_colors
MPE_Num_colors — Gets the number of available colors	

Synopsis

```
#include "mpe.h"
int MPE_Num_colors( handle, nc )
MPE_XGraph handle;
int *nc;
```

Input Parameter

handle MPE graphics handle

Output Parameter

nc Number of colors available on the display.

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, ierr, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.  
All MPI and MPE objects, MPI\_Comm, MPE\_XGraph and MPE\_Color, are of type INTEGER in Fortran.

Location

./mpe/src/mpe\_graphics.c

MPE_Open_graphics	MPE_Open_graphics
MPE_Open_graphics — (collectively) opens an X Windows display	

Synopsis

```
#include "mpe.h"
int MPE_Open_graphics( handle, comm, display, x, y, w, h, is_collective )
MPE_XGraph *handle;
MPI_Comm comm;
char display[MPI_MAX_PROCESSOR_NAME+1];
int x, y;
int w, h;
int is_collective;
```

Input Parameters

- comm** Communicator of participating processes
- display** Name of X window display. If null, display will be taken from the DISPLAY variable on the process with rank 0 in **comm**. If that is either undefined, or starts with `w":p"`, then the value of display is `'hostname':0`
- x,y** position of the window. If `(-1,-1)`, then the user should be asked to position the window (this is a window manager issue).
- w,h** width and height of the window, in pixels.
- is\_collective** true if the graphics operations are collective; this allows the MPE graphics operations to make fewer connections to the display. If false, then all processes in the communicator **comm** will open the display; this could exceed the number of connections that your X window server allows. Not yet implemented.

Output Parameter

- handle** Graphics handle to be given to other MPE graphics routines.

Notes

This is a collective routine. All processes in the given communicator must call it, and it has the same semantics as **MPI\_Barrier** (that is, other collective operations can not cross this routine).

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, **ierr**, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the CALL statement.

All MPI and MPE objects, **MPI\_Comm**, **MPE\_XGraph** and **MPE\_Color**, are of type **INTEGER** in Fortran.

Additional Notes for Fortran Interface

If Fortran **display** argument is an empty string, "", display will be taken from the **DISPLAY** variable on the process with rank 0 in **comm**. The trailing blanks in Fortran **CHARACTER** string argument will be ignored.

Location

./mpe/src/mpe\_graphics.c

MPE_Print_datatype_pack_action	MPE_Print_datatype_pack_action
MPE_Print_datatype_pack_action — Prints the operations performed in an pack of a datatype	

Synopsis

```
int MPE_Print_datatype_pack_action( fp, count, type, in_offset, out_offset )
FILE *fp;
int count;
MPI_Datatype type;
int in_offset, out_offset;
```

Input Parameters

- fp** FILE pointer for output
- count** Count of datatype
- type** MPI\_Datatype
- in\_offset,out\_offset** offsets for input and output buffer. Should be 0 for most uses.

Notes

This prints on the selected file the operations that the MPICH implementation will take when packing a buffer

Location

./mpe/src/examine.c

MPE_Print_datatype_unpack_action	MPE_Print_datatype_unpack_action
MPE_Print_datatype_unpack_action — Prints the operations performed in an unpack of a datatype	

Synopsis

```
int MPE_Print_datatype_unpack_action( fp, count, type, in_offset, out_offset )
FILE *fp;
int count;
MPI_Datatype type;
int in_offset, out_offset;
```

Input Parameters

**fp** FILE pointer for output  
**count** Count of datatype  
**type** MPI Datatype  
**in\_offset,out\_offset** offsets for input and output buffer. Should be 0 for most uses.

Notes

This prints on the selected file the operations that the MPICH implementation will take when unpacking a buffer.

Location

./mpe/src/examine.c

MPE_ReturnTags		MPE_ReturnTags
MPE_ReturnTags — Returns tags allocated with MPE_GetTags.		
Synopsis		
int MPE_ReturnTags( comm, first_tag, ntags ) MPI_Comm comm; int first_tag, ntags;		
Input Parameters		
comm	Communicator to return tags to	
first_tag	First of the tags to return	
ntags	Number of tags to return.	
See Also		
MPE_GetTags		
Location		
./mpe/src/privtags.c		
MPE_Seq_begin		MPE_Seq_begin
MPE_Seq_begin — Begins a sequential section of code.		

Synopsis

void MPE\_Seq\_begin( MPI\_Comm comm, int ng )

Input Parameters

**comm** Communicator to sequentialize.  
**ng** Number in group. This many processes are allowed to execute at the same time. Usually one.

Notes

MPE\_Seq\_begin and MPE\_Seq\_end provide a way to force a section of code to be executed by the processes in rank order. Typically, this is done with

MPE\_Seq\_begin( comm, 1 );  
<code to be executed sequentially>  
MPE\_Seq\_end( comm, 1 );

Often, the sequential code contains output statements (e.g., printf) to be executed. Note that you may need to flush the I/O buffers before calling MPE\_Seq\_end; also note that some systems do not propagate I/O in any order to the controlling terminal (in other words, even if you flush the output, you may not get the data in the order that you want).

Location

./mpe/src/mpe\_seq.c

MPE_Seq_end		MPE_Seq_end
MPE_Seq_end — Ends a sequential section of code.		
Synopsis		
void MPE_Seq_end( MPI_Comm comm, int ng )		
Input Parameters		
comm	Communicator to sequentialize.	
ng	Number in group. This many processes are allowed to execute at the same time. Usually one.	
Notes		
See MPE_Seq_begin for more details.		

Location

./mpe/src/mpe\_seq.c

MPE\_Start\_log

MPE\_Start\_log

MPE\_Start\_log — Begin logging of events

Synopsis

int MPE\_Start\_log()

Location

./mpe/src/mpe\_log.c

MPE\_Stop\_log

MPE\_Stop\_log

MPE\_Stop\_log — Stop logging events

Synopsis

int MPE\_Stop\_log()

Location

./mpe/src/mpe\_log.c

MPE\_TagsEnd

MPE\_TagsEnd

MPE\_TagsEnd — Returns the private keyval.

Synopsis

int MPE\_TagsEnd()

Notes

This routine is provided to aid in cleaning up all of the allocated storage in and MPI program. Normally, this routine does *not* need to be called. If it is, it should be called immediately before `MPI_Finalize`.

Location

./mpe/src/privtags.c

MPE\_Update

MPE\_Update

MPE\_Update — Updates an X11 display

Synopsis

```
#include "mpe.h"
int MPE_Update( handle )
MPE_XGraph handle;
```

Input Parameter

handle      MPE graphics handle.

Note

Only after an `MPE_Update` can you count on seeing the results of MPE drawing routines. This is caused by the buffering of graphics requests for improved performance.

Notes For Fortran Interface

The Fortran interface to this routine is different from its C counterpart and it has an additional argument, `ierr`, at the end of the argument list, i.e. the returned function value (the error code) in C interface is returned as the additional argument in Fortran interface. The Fortran interface is invoked with the `CALL` statement.  
All MPI and MPE objects, `MPI_Comm`, `MPE_XGraph` and `MPE_Color`, are of type `INTEGER` in Fortran.

Location

./mpe/src/mpe\_graphics.c