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

February 1, 1996

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 **92**

4 ADI routines **112**

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

MPI_Abort **MPI_Abort**

MPI_Abort — Terminates MPI execution environment

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.

Synopsis

```
int MPI_Abort( comm, errorcode )
MPI_Comm         comm;
int              errorcode;
```

Location

`src/env/abort.c`

MPI_Address **MPI_Address**

MPI_Address — Gets the address of a location in memory

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.

Synopsis

```
int MPI_Address( location, address)
void     *location;
MPI_Aint *address;
```

Location

`src/pt2pt/address.c`

MPI_Allgather **MPI_Allgather**

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

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)
recvcount 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 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.

Synopsis

```
int MPI_Allgather ( sendbuf, sendcount, sendtype,
                    recvbuf, recvcount, recvtype, comm )
void             *sendbuf;
int               sendcount;
MPI_Datatype      sendtype;
void             *recvbuf;
int               recvcount;
MPI_Datatype      recvtype;
MPI_Comm          comm;
```

Location

src/coll/allgather.c

MPI_Allgatherv **MPI_Allgatherv**

MPI_Allgatherv — Gathers data from all tasks and deliver it to all

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`
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.

Synopsis

```
int MPI_Allgatherv ( sendbuf, sendcount,  sendtype,
                     recvbuf, recvcounts, displs,   recvtype, comm )
void             *sendbuf;
int               sendcount;
MPI_Datatype      sendtype;
void             *recvbuf;
int              *recvcounts;
```

```
int              *displs;
MPI_Datatype      recvtype;
MPI_Comm          comm;
```

Location

src/coll/allgatherv.c

MPI_Allreduce **MPI_Allreduce**

MPI_Allreduce — Combines values from all processes and distribute the result back to all processes

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 `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.

Synopsis

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

Location

src/coll/allreduce.c

MPI_Alltoall **MPI_Alltoall**

MPI_Alltoall — Sends data from all to all processes

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)
recvcount 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 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.

Synopsis

```
int MPI_Alltoall( sendbuf, sendcount, sendtype,
                  recvbuf, recvcnt, recvtype, comm )
void             *sendbuf;
int               sendcount;
MPI_Datatype      sendtype;
void             *recvbuf;
int               recvcnt;
MPI_Datatype      recvtype;
MPI_Comm          comm;
```

Location

`src/coll/alltoall.c`

MPI_Alltoallv — Sends data from all to all processes, with a displacement

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.

Synopsis

```
int MPI_Alltoallv ( sendbuf, sendcnts, sdispls, sendtype,
                    recvbuf, recvcnts, rdispls, recvtype, comm )
void             *sendbuf;
int              *sendcnts;
int              *sdispls;
MPI_Datatype      sendtype;
void             *recvbuf;
int              *recvcnts;
int              *rdispls;
MPI_Datatype      recvtype;
MPI_Comm          comm;
```

Location

`src/coll/alltoallv.c`

MPI_Attr_delete **MPI_Attr_delete**

MPI_Attr_delete — Deletes attribute value associated with a key

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.

Synopsis

```
int MPI_Attr_delete ( comm, keyval )
MPI_Comm comm;
int      keyval;
```

Location

`src/context/attr_delval.c`

MPI_Attr_get **MPI_Attr_get**

MPI_Attr_get — Retrieves attribute value by key

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` arguement 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 *`.

Synopsis

```
int MPI_Attr_get ( comm, keyval, attr_value, flag )
MPI_Comm comm;
int keyval;
void *attr_value;
int *flag;
```

Location

`src/context/attr_getval.c`

MPI_Attr_put **MPI_Attr_put**

MPI_Attr_put — Stores attribute value associated with a key

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.

See Also

MPI_Attr_get, MPI_Keyval_create, MPI_Attr_delete

Synopsis

```
int MPI_Attr_put ( comm, keyval, attr_value )
MPI_Comm comm;
int      keyval;
void     *attr_value;
```

Location

`src/context/attr_putval.c`

MPI_Barrier **MPI_Barrier**

MPI_Barrier — Blocks until all process have reached this routine.

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 modifiy 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.

Synopsis

```
int MPI_Barrier ( comm )
MPI_Comm comm;
```

Location

`src/coll/barrier.c`

MPI_Bcast **MPI_Bcast**

MPI_Bcast — Broadcasts a message from the process with rank "root" to all other processes of the group.

Input/output Parameters

buffer starting address of buffer (choice)
count number of entries in buffer (integer)
datatype data type of buffer (handle)
root rank of broadcast root (integer)
comm 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.

Synopsis

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

Location

`src/coll/bcast.c`

MPI_Bsend **MPI_Bsend**

MPI_Bsend — Basic send with user-specified buffering

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

The buffer space can not be reused unless you are certain that the message has been received (not just that it should have been received). 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.

See Also

MPI_Buffer_attach, MPI_Ibsend, MPI_Bsend_init

Synopsis

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

Location

`src/pt2pt/bsend.c`

MPI_Bsend_init **MPI_Bsend_init**

MPI_Bsend_init — Builds a handle for a buffered send

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.

Synopsis

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

Location

src/pt2pt/bsend_init.c

MPI_Buffer_attach **MPI_Buffer_attach**

MPI_Buffer_attach — Attaches a user-defined buffer for sending

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 Bsends 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.

Synopsis

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

Location

src/pt2pt/bufattach.c

MPI_Buffer_detach **MPI_Buffer_detach**

MPI_Buffer_detach — Removes an existing buffer (for use in MPI_Bsend etc)

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).

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.

Synopsis

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

Location

`src/pt2pt/buffree.c`

MPI_Cancel **MPI_Cancel**

MPI_Cancel — Cancels a communication request

Input Parameter

request communication request (handle)

Note

Cancel has not been implemented yet.

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.

Synopsis

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

Location

`src/pt2pt/cancel.c`

MPI_Cart_coords **MPI_Cart_coords**

MPI_Cart_coords — Determines process coords in cartesian topology given rank in group

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.

Synopsis

```
int MPI_Cart_coords ( comm, rank, maxdims, coords )
MPI_Comm  comm;
int       rank;
int       maxdims;
```

```
int      *coords;
```

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

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.

Synopsis

```
int MPI_Cart_create ( comm_old, ndims, dims, periods, reorder, comm_cart )
MPI_Comm  comm_old;
int       ndims;
int      *dims;
int      *periods;
int       reorder;
MPI_Comm *comm_cart;
```

Location

src/topol/cart_create.c

MPI_Cart_get **MPI_Cart_get**

MPI_Cart_get — Retrieves Cartesian topology information associated with a communicator

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.

Synopsis

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

Location

`src/topol/cart_get.c`

MPI_Cart_map — Maps process to Cartesian topology information

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.

Synopsis

```
int MPI_Cart_map ( comm_old, ndims, dims, periods, newrank )
MPI_Comm comm_old;
int      ndims;
int     *dims;
int     *periods;
int     *newrank;
```

Location

src/topol/cart_map.c

MPI_Cart_rank **MPI_Cart_rank**

MPI_Cart_rank — Determines process rank in communicator given Cartesian location

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 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.

Synopsis

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

Location

src/topol/cart_rank.c

MPI_Cart_shift **MPI_Cart_shift**

MPI_Cart_shift — Returns the shifted source and destination ranks, given a shift direction and amount

Input Parameters

comm communicator with cartesian structure (handle)
direction coordinate dimension of shift (integer)
disp 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.

Synopsis

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

Location

`src/topol/cart_shift.c`

MPI_Cart_sub **MPI_Cart_sub**

MPI_Cart_sub — Partitions a communicator into subgroups which form lower-dimensional cartesian subgrids

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.

Synopsis

```
int MPI_Cart_sub ( comm, remain_dims, comm_new )
MPI_Comm  comm;
int      *remain_dims;
MPI_Comm *comm_new;
```

Location

`src/topol/cart_sub.c`

MPI_Cartdim_get **MPI_Cartdim_get**

MPI_Cartdim_get — Retrieves Cartesian topology information associated with a communicator

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.

Synopsis

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

Location

`src/topol/cartdim_get.c`

MPI_Comm_compare **MPI_Comm_compare**

MPI_Comm_compare — Compares two communicators

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

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.

Synopsis

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

Location

`src/context/commcompare.c`

MPI_Comm_create **MPI_Comm_create**

MPI_Comm_create — Creates a new communicator

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.

Synopsis

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

Location

```
src/context/comm_create.c
```

MPI_Comm_dup **MPI_Comm_dup**

MPI_Comm_dup — Duplicates an existing communicator with all its cached information

Input Parameter

comm communicator (handle)

Output Parameter

newcomm copy of `comm` (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.

Synopsis

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

Location

```
src/context/comm_dup.c
```

MPI_Comm_free **MPI_Comm_free**

MPI_Comm_free — Marks the communicator object for deallocation

Input Parameter

comm communicator to be destroyed (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.

Synopsis

```
int MPI_Comm_free ( commp )
MPI_Comm *commp;
```

Location

`src/context/comm_free.c`

MPI_Comm_group **MPI_Comm_group**

MPI_Comm_group — Accesses the group associated with given communicator

Input Parameter

comm Communicator

Output Parameter

group Group in 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.

Synopsis

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

Location

`src/context/comm_group.c`

MPI_Comm_rank **MPI_Comm_rank**

MPI_Comm_rank — Determines the rank of the calling process in the communicator

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.

Synopsis

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

Input Parameter

comm Communicator

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.

Synopsis

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

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 intercommunictor

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.

Synopsis

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

Location

src/context/comm_rsize.c

MPI_Comm_size **MPI_Comm_size**

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

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.

Synopsis

```
int MPI_Comm_size ( comm, size )
MPI_Comm comm;
int *size;
```

Location

`src/context/comm_size.c`

MPI_Comm_split **MPI_Comm_split**

MPI_Comm_split — Creates new communicators based on colors and keys

Input Parameters

comm	communicator (handle)
color	control of subset assignment (nonnegative integer)
key	control of rank assigment (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 '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 'MPI_UNDEFINED', it can return
    a '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.
```

Synopsis

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

Location

`src/context/comm_split.c`

MPI_Comm_test_inter **MPI_Comm_test_inter**

MPI_Comm_test_inter — Tests to see if a comm is an inter-communicator

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.

Synopsis

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

Location

`src/context/comm_testic.c`

MPI_DUP_FN **MPI_DUP_FN**

MPI_DUP_FN — A function to simple-mindedly copy attributes

Synopsis

```
int MPIR_dup_fn ( comm, keyval, extra_state, attr_in, attr_out, flag )
MPI_Comm  comm;
int       keyval;
void      *extra_state;
void      *attr_in;
void      *attr_out;
int       *flag;
```

Location

src/context/dup_fn.c

MPI_Dims_create **MPI_Dims_create**

MPI_Dims_create — Creates a division of processors in a cartesian grid

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.

Synopsis

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

Location

src/topol/dims_create.c

MPI_Errhandler_create **MPI_Errhandler_create**

MPI_Errhandler_create — Creates an MPI-style errorhandler

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.

Synopsis

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

Location

`src/env/errcreate.c`

MPI_Errhandler_free **MPI_Errhandler_free**

MPI_Errhandler_free — Frees an MPI-style errorhandler

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.

Synopsis

```
int MPI_Errhandler_free( errhandler )
MPI_Errhandler *errhandler;
```

Location

src/env/errfree.c

MPI_Errhandler_get **MPI_Errhandler_get**

MPI_Errhandler_get — Gets the error handler for a communicator

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.

Synopsis

```
int MPI_Errhandler_get( comm, errhandler )
MPI_Comm comm;
MPI_Errhandler *errhandler;
```

Location

src/env/errget.c

MPI_Errhandler_set **MPI_Errhandler_set**

MPI_Errhandler_set — Sets the error handler for a communicator

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.

Synopsis

```
int MPI_Errhandler_set( comm, errhandler )
MPI_Comm       comm;
MPI_Errhandler errhandler;
```

Location

`src/env/errset.c`

MPI_Error_class **MPI_Error_class**

MPI_Error_class — Converts an error code into an error class

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.

Synopsis

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

Location

`src/env/errclass.c`

MPI_Error_string **MPI_Error_string**

MPI_Error_string — Return a string for a given error code

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.

Synopsis

```
int MPI_Error_string( errorcode, string, resultlen )
int  errorcode, *resultlen;
char *string;
```

Location

`src/env/errorstring.c`

MPI_Finalize **MPI_Finalize**

MPI_Finalize — Terminates MPI execution environment

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.

Synopsis

```
int MPI_Finalize()
```

Location

`src/env/finalize.c`

MPI_Gather **MPI_Gather**

MPI_Gather — Gathers together values from a group of processes

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)
recvcount	number of elements for any single receive (integer, significant only at root)
recvtype	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.

Synopsis

```
int MPI_Gather ( sendbuf, sendcnt, sendtype, recvbuf, recvcount, recvtype,
   root, comm )
void             *sendbuf;
int               sendcnt;
MPI_Datatype      sendtype;
void             *recvbuf;
int               recvcount;
MPI_Datatype      recvtype;
int               root;
MPI_Comm          comm;
```

Location

`src/coll/gather.c`

MPI_Gatherv **MPI_Gatherv**

MPI_Gatherv — Gathers into specified locations from all processes in a group

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)
recvtype	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.

Synopsis

```
int MPI_Gatherv ( sendbuf, sendcnt,  sendtype,
                  recvbuf, recvcnts, displs, recvtype,
                  root, comm )
void             *sendbuf;
int               sendcnt;
MPI_Datatype      sendtype;
void             *recvbuf;
int              *recvcnts;
int              *displs;
MPI_Datatype      recvtype;
int               root;
MPI_Comm          comm;
```

Location

`src/coll/gatherv.c`

MPI_Get_count **MPI_Get_count**

MPI_Get_count — Gets the number of "top level" elements

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.

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.

Synopsis

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

Location

`src/pt2pt/getcount.c`

MPI_Get_elements **MPI_Get_elements**

MPI_Get_elements — Returns the number of basic elements in a datatype

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.

Synopsis

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

Location

`src/pt2pt/getelements.c`

MPI_Get_processor_name **MPI_Get_processor_name**

MPI_Get_processor_name — Gets the name of the processor

Output Parameters

name	A unique specifier for the actual (as opposed to virtual) node.
resultlen	Length (in characters) of the name

Notes

The name returned should identify a particular piece of hardware; the exact format is implementation defined. This name may or may not be the same as might be returned by `gethostname`, `uname`, or `sysinfo`.

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.

Synopsis

```
int MPI_Get_processor_name( name, resultlen )
char *name;
int *resultlen;
```

Location

`src/env/getpname.c`

MPI_Graph_create **MPI_Graph_create**

MPI_Graph_create — Makes a new communicator to which topology information has been attached

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.

Synopsis

```
int MPI_Graph_create ( comm_old, nnodes, index, edges, reorder, comm_graph )
MPI_Comm  comm_old;
int       nnodes;
int      *index;
int      *edges;
int       reorder;
MPI_Comm *comm_graph;
```

Location

`src/topol/graphcreate.c`

MPI_Graph_get **MPI_Graph_get**

MPI_Graph_get — Retrieves graph topology information associated with a communicator

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.

Synopsis

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

Location

src/topol/graph_get.c

MPI_Graph_map **MPI_Graph_map**

MPI_Graph_map — Maps process to graph topology information

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.

Synopsis

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

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

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.

Synopsis

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

Location

`src/topol/graph_nbr.c`

MPI_Graph_neighbors_count **MPI_Graph_neighbors_count**

MPI_Graph_neighbors_count — Returns the number of neighbors of a node associated with a graph topology

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.

Synopsis

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

Location

src/topol/graphnbrcnt.c

MPI_Graphdims_get **MPI_Graphdims_get**

MPI_Graphdims_get — Retrieves graph topology information associated with a communicator

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.

Synopsis

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

Location

src/topol/graphdimsget.c

MPI_Group_compare **MPI_Group_compare**

MPI_Group_compare — Compares two groups

Input Parameters

group1 group1 (handle)
group2 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.

Synopsis

```
int MPI_Group_compare ( group1, group2, result )
MPI_Group  group1;
MPI_Group  group2;
int       *result;
```

Location

`src/context/groupcompare.c`

MPI_Group_difference **MPI_Group_difference**

MPI_Group_difference — Makes a group from the difference of two groups

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.

Synopsis

```
int MPI_Group_difference ( group1, group2, group_out )
MPI_Group group1, group2, *group_out;
```

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

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.

Synopsis

```
int MPI_Group_excl ( group, n, ranks, newgroup )
MPI_Group group, *newgroup;
int       n, *ranks;
```

Location

src/context/group_excl.c

MPI_Group_free **MPI_Group_free**

MPI_Group_free — Frees a 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.

Synopsis

```
int MPI_Group_free ( group )
MPI_Group *group;
```

Location

`src/context/group_free.c`

MPI_Group_incl **MPI_Group_incl**

MPI_Group_incl — Produces a group by reordering an existing group and taking only listed members

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 `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.

Synopsis

```
int MPI_Group_incl ( group, n, ranks, group_out )
MPI_Group group, *group_out;
int       n, *ranks;
```

Location

src/context/group_incl.c

MPI_Group_intersection **MPI_Group_intersection**

MPI_Group_intersection — Produces a group as the intersection of two existing groups

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 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.

Synopsis

```
int MPI_Group_intersection ( group1, group2, group_out )
MPI_Group group1, group2, *group_out;
```

Location

src/context/group_inter.c

MPI_Group_range_excl **MPI_Group_range_excl**

MPI_Group_range_excl — Produces a group by excluding ranges of processes from an existing group

Input Parameters

group group (handle)
n number of elements in array ranks (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.

Synopsis

```
int MPI_Group_range_excl ( group, n, ranges, newgroup )
MPI_Group group, *newgroup;
int       n, ranges[][3];
```

Location

`src/context/group_rexcl.c`

MPI_Group_range_incl — Creates a new group from ranges of ranks in an existing group

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.

Synopsis

```
int MPI_Group_range_incl ( group, n, ranges, newgroup )
MPI_Group group, *newgroup;
int       n, ranges[][3];
```

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

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.

Synopsis

```
int MPI_Group_rank ( group, rank )
MPI_Group group;
int *rank;
```

Location

`src/context/group_rank.c`

MPI_Group_size **MPI_Group_size**

MPI_Group_size — Returns the size of a group

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 `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.

Synopsis

```
int MPI_Group_size ( group, size )
MPI_Group group;
int *size;
```

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

Input Parameters

group1	group1 (handle)
n	number of ranks in `ranks1` and `ranks2` arrays (integer)
ranks1	array of zero or more valid ranks in `group1`
group2	group2 (handle)

Output Parameter

ranks2	array of corresponding ranks in group2, `MPI_UNDEFINED` when no correspondence exists.

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.

Synopsis

```
int MPI_Group_translate_ranks ( group_a, n, ranks_a, group_b, ranks_b )
MPI_Group group_a;
int n;
int       *ranks_a;
MPI_Group  group_b;
int       *ranks_b;
```

Location

src/context/group_tranks.c

MPI_Group_union **MPI_Group_union**

MPI_Group_union — Produces a group by combining two groups

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.

Synopsis

```
int MPI_Group_union ( group1, group2, group_out )
MPI_Group group1, group2, *group_out;
```

Location

src/context/group_union.c

MPI_Ibsend **MPI_Ibsend**

MPI_Ibsend — Starts a nonblocking buffered send

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.

Synopsis

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

Location

`src/pt2pt/ibsend.c`

MPI_Init **MPI_Init**

MPI_Init — Initialize the MPI execution environment

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`.

-mpipktsize nn - Set the message length where the ADI changed to the long message protocol to `nn`. This only works if MPICH was configured with `-var_pkt`.

The following options are available only on the Chameleon device and devices built with debugging code. Normally, these should only be used by MPICH implementors when debugging new ports.

-mpichdebug Print out the Chameleon device operations

-mpidbfile filename - Like mpichdebug, but sends the output to the specified file. If the filename contains a `%d`, then that part of the filename is replaced with the rank in `MPI_COMM_WORLD`. For example,

```
-mpidbfile log.%d
```

writes to `log.0`, `log.1`, etc.

-mpichmemdebug (Chameleon device only) Print out a list of unreclaimed memory. This requires that MPI be built with the `-DMPIR_DEBUG_MEM` switch. This is intended for debugging the MPI implementation itself.

-mpimem If MPICH was built with `-DMPIR_DEBUG_MEM`, this checks all malloc and free operations (internal to MPICH) for signs of injury to the memory allocation areas.

-mpichmsg Print out the number of messages received, by category, when the program exits.

Notes

Note that the Fortran binding for this routine has only the error return argument (`MPI_INIT(ierror)`)

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 truely 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 independantly 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

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.

Synopsis

```
int MPI_Init(argc,argv)
int  *argc;
char ***argv;
```

Location

`src/env/init.c`

MPI_Initialized **MPI_Initialized**

MPI_Initialized — Indicates whether `MPI_Init` has been called.

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.

Synopsis

```
int MPI_Initialized( flag )
int  *flag;
```

Location

`src/env/initialize.c`

MPI_Intercomm_create **MPI_Intercomm_create**

MPI_Intercomm_create — Creates an intercommuncator from two intracommunicators

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 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 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 inter-coll context, and an intra-coll context
2) Send "send_context" and lrank_to_grank list from local comm group if Im the local_leader.
3) If Im 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
```

Synopsis

```
int MPI_Intercomm_create ( local_comm, local_leader, peer_comm,
                           remote_leader, tag, comm_out )
MPI_Comm  local_comm;
int       local_leader;
MPI_Comm  peer_comm;
int       remote_leader;
int       tag;
MPI_Comm *comm_out;
```

Location

`src/context/ic_create.c`

MPI_Intercomm_merge **MPI_Intercomm_merge**

MPI_Intercomm_merge — Creates an intracommuncator from an intercommunicator

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

Synopsis

```
int MPI_Intercomm_merge ( comm, high, comm_out )
MPI_Comm  comm;
int       high;
MPI_Comm *comm_out;
```

Location

`src/context/ic_merge.c`

MPI_Iprobe **MPI_Iprobe**

MPI_Iprobe — Nonblocking test for a message

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.

Synopsis

```
int MPI_Iprobe( source, tag, comm, flag, status )
int         source;
int         tag;
int         *flag;
MPI_Comm    comm;
MPI_Status  *status;
```

Location

`src/pt2pt/iprobe.c`

MPI_Irecv **MPI_Irecv**

MPI_Irecv — Begins a nonblocking receive

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.

Synopsis

```
int MPI_Irecv( buf, count, datatype, source, tag, comm, request )
void             *buf;
int              count;
MPI_Datatype     datatype;
int              source;
int              tag;
MPI_Comm         comm;
MPI_Request      *request;
```

Location

`src/pt2pt/irecv.c`

MPI_Irsend **MPI_Irsend**

MPI_Irsend — Starts a nonblocking ready send

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.

Synopsis

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

```
MPI_Request      *request;
```

Location

src/pt2pt/irsend.c

MPI_Isend **MPI_Isend**

MPI_Isend — Begins a nonblocking send

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.

Synopsis

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

Location

src/pt2pt/isend.c

MPI_Issend **MPI_Issend**

MPI_Issend — Starts a nonblocking synchronous send

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.

Synopsis

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

Location

`src/pt2pt/issend.c`

MPI_Keyval_create **MPI_Keyval_create**

MPI_Keyval_create — Generates a new attribute key

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 programers 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.

Synopsis

```
int MPI_Keyval_create ( copy_fn, delete_fn, keyval, extra_state )
MPI_Copy_function   *copy_fn;
MPI_Delete_function *delete_fn;
int                 *keyval;
void                *extra_state;
```

Location

`src/context/keyvalcreate.c`

MPI_Keyval_free **MPI_Keyval_free**

MPI_Keyval_free — Frees attribute key for communicator cache attribute

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.

Synopsis

```
int MPI_Keyval_free ( keyval )
int *keyval;
```

Location

`src/context/keyval_free.c`

MPI_NULL_COPY_FN **MPI_NULL_COPY_FN**

MPI_NULL_COPY_FN — A function to not copy attributes

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

Input Parameters

function	user defined function (function)
commute	true if commutative; false otherwise.

Output Parameter

op	operation (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.

Synopsis

```
int MPI_Op_create( function, commute, op )
MPI_User_function *function;
int               commute;
MPI_Op            *op;
```

Location

```
src/coll/opcreate.c
```

MPI_Op_free **MPI_Op_free**

MPI_Op_free — Frees a user-defined combination function handle

Input Parameter

op operation (handle)

Notes

`op` is set to `MPI_OP_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.

Synopsis

```
int MPI_Op_free( op )
MPI_Op  *op;
```

Location

```
src/coll/opfree.c
```

MPI_Pack **MPI_Pack**

MPI_Pack — Packs a datatype into contiguous memory

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.

See Also

MPI_Unpack, MPI_Pack_size

Synopsis

```
int MPI_Pack ( inbuf, incount, type, outbuf, outcount, position, comm )
void         *inbuf;
int           incount;
MPI_Datatype  type;
void         *outbuf;
int           outcount;
int          *position;
MPI_Comm      comm;
```

Location

`src/pt2pt/pack.c`

MPI_Pack_size **MPI_Pack_size**

MPI_Pack_size — Returns the upper bound on the amount of space needed to pack a message

Input Parameters

incount 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 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.

Synopsis

```
int MPI_Pack_size ( incount, datatype, comm, size )
int           incount;
MPI_Datatype  datatype;
MPI_Comm      comm;
int          *size;
```

Location

src/pt2pt/pack_size.c

MPI_Pcontrol **MPI_Pcontrol**

MPI_Pcontrol — Controls profiling

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.

Synopsis

```
int MPI_Pcontrol( level )
int level;
```

Location

src/profile/pcontrol.c

MPI_Probe **MPI_Probe**

MPI_Probe — Blocking test for a message

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.

Synopsis

```
int MPI_Probe( source, tag, comm, status )
int         source;
int         tag;
MPI_Comm    comm;
MPI_Status  *status;
```

Location

`src/pt2pt/probe.c`

MPI_Recv **MPI_Recv**

MPI_Recv — Basic receive

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.

Synopsis

```
int MPI_Recv( buf, count, datatype, source, tag, comm, status )
void            *buf;
int             count, source, tag;
MPI_Datatype    datatype;
MPI_Comm        comm;
MPI_Status      *status;
```

Location

src/pt2pt/recv.c

MPI_Recv_init **MPI_Recv_init**

MPI_Recv_init — Builds a handle for a receive

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.

Synopsis

```
int MPI_Recv_init( buf, count, datatype, source, tag, comm, request )
void         *buf;
int          count;
MPI_Request  *request;
MPI_Datatype datatype;
int          source;
int          tag;
MPI_Comm     comm;
```

Location

src/pt2pt/create_recv.c

MPI_Reduce **MPI_Reduce**

MPI_Reduce — Reduces values on all processes to a single value

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.

Synopsis

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

Location

`src/coll/reduce.c`

MPI_Reduce_scatter **MPI_Reduce_scatter**

MPI_Reduce_scatter — Combines values and scatters the results

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.

Synopsis

```
int MPI_Reduce_scatter ( sendbuf, recvbuf, recvcnts, datatype, op, comm )
void             *sendbuf;
void             *recvbuf;
int              *recvcnts;
MPI_Datatype      datatype;
MPI_Op            op;
MPI_Comm          comm;
```

Location

`src/coll/red_scat.c`

MPI_Request_free — Frees a communication request object

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.

Synopsis

```
int MPI_Request_free( request )
MPI_Request *request;
```

Location

src/pt2pt/commreq_free.c

MPI_Rsend **MPI_Rsend**

MPI_Rsend — Basic ready send

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.

Synopsis

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

Location

src/pt2pt/rsend.c

MPI_Rsend_init **MPI_Rsend_init**

MPI_Rsend_init — Builds a handle for a ready send

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)

Synopsis

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

Location

`src/pt2pt/rsend_init.c`

MPI_Scan **MPI_Scan**

MPI_Scan — Computes the scan (partial reductions) of data on a collection of processes

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.

Synopsis

```
int MPI_Scan ( sendbuf, recvbuf, count, datatype, op, comm )
void             *sendbuf;
void             *recvbuf;
int               count;
```

```
MPI_Datatype      datatype;
MPI_Op            op;
MPI_Comm          comm;
```

Location

src/coll/scan.c

MPI_Scatter **MPI_Scatter**

MPI_Scatter — Sends data from one task to all other tasks in a group

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.

Synopsis

```
int MPI_Scatter ( sendbuf, sendcnt, sendtype,
    recvbuf, recvcnt, recvtype,
    root, comm )
void             *sendbuf;
int               sendcnt;
MPI_Datatype      sendtype;
void             *recvbuf;
int               recvcnt;
MPI_Datatype      recvtype;
int               root;
MPI_Comm          comm;
```

Location

src/coll/scatter.c

MPI_Scatterv **MPI_Scatterv**

MPI_Scatterv — Scatters a buffer in parts to all tasks in a group

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.

Synopsis

```
int MPI_Scatterv ( sendbuf, sendcnts, displs, sendtype,
                   recvbuf, recvcnt,  recvtype,
                   root, comm )
void             *sendbuf;
int              *sendcnts;
int              *displs;
MPI_Datatype      sendtype;
void             *recvbuf;
int               recvcnt;
MPI_Datatype      recvtype;
int               root;
MPI_Comm          comm;
```

Location

`src/coll/scatterv.c`

MPI_Send **MPI_Send**

MPI_Send — Performs a basic send

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.

See Also

MPI_Isend, MPI_Bsend

Synopsis

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

Location

`src/pt2pt/send.c`

MPI_Send_init **MPI_Send_init**

MPI_Send_init — Builds a handle for a standard send

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.

Synopsis

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

Location

`src/pt2pt/create_send.c`

MPI_Sendrecv — Sends and receives a message

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.

Synopsis

```
int MPI_Sendrecv( sendbuf, sendcount, sendtype, dest, sendtag,
                  recvbuf, recvcount, recvtype, source, recvtag,
                  comm, status )
void         *sendbuf;
int           sendcount;
MPI_Datatype  sendtype;
int           dest, sendtag;
void         *recvbuf;
int           recvcount;
MPI_Datatype  recvtype;
int           source, recvtag;
MPI_Comm      comm;
MPI_Status   *status;
```

Location

```
src/pt2pt/sendrecv.c
```

MPI_Sendrecv_replace **MPI_Sendrecv_replace**

MPI_Sendrecv_replace — Sends and receives using a single buffer

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.

Synopsis

```
int MPI_Sendrecv_replace( buf, count, datatype, dest, sendtag,
     source, recvtag, comm, status )
void         *buf;
int           count, dest, sendtag, source, recvtag;
```

```
MPI_Datatype  datatype;
MPI_Comm      comm;
MPI_Status   *status;
```

Location

`src/pt2pt/sendrecv_rep.c`

MPI_Ssend **MPI_Ssend**

MPI_Ssend — Basic synchronous send

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.

Synopsis

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

Location

`src/pt2pt/ssend.c`

MPI_Ssend_init **MPI_Ssend_init**

MPI_Ssend_init — Builds a handle for a synchronous send

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)

Synopsis

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

Location

src/pt2pt/ssend_init.c

MPI_Start **MPI_Start**

MPI_Start — Initiates a communication with a persistent request handle

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.

Synopsis

```
int MPI_Start( request )
MPI_Request *request;
```

Location

src/pt2pt/start.c

MPI_Startall **MPI_Startall**

MPI_Startall — Starts a collection 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.

Synopsis

```
int MPI_Startall( count, array_of_requests )
int count;
MPI_Request array_of_requests[];
```

Location

`src/pt2pt/startall.c`

MPI_Test **MPI_Test**

MPI_Test - Tests for the completion of a send or receive

Input Parameter

request communication request (handle)

Output Parameter

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.

Synopsis

```
int MPI_Test ( request, flag, status )
MPI_Request  *request;
int          *flag;
MPI_Status   *status;
```

Location

`src/pt2pt/test.c`

MPI_Test_cancelled **MPI_Test_cancelled**

MPI_Test_cancelled — Tests to see if a request was canceled

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.

Synopsis

```
int MPI_Test_cancelled( status, flag )
MPI_Status *status;
int        *flag;
```

Location

`src/pt2pt/testcancel.c`

MPI_Testall **MPI_Testall**

MPI_Testall — Tests for the completion of all previously initiated communications

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)

Notes

`flag` is true only if all requests ave completed. Otherwise, flag is false and neither the `array_of_requests` nor the `array_of_statuses` is modified.

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.

Synopsis

```
int MPI_Testall( count, array_of_requests, flag, array_of_statuses )
int        count;
MPI_Request array_of_requests[];
int        *flag;
MPI_Status *array_of_statuses;
```

Location

`src/pt2pt/testall.c`

MPI_Testany **MPI_Testany**

MPI_Testany — Tests for completion of any previdously initiated communication

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)

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.

Synopsis

```
int MPI_Testany( count, array_of_requests, index, flag, status )
int         count;
MPI_Request array_of_requests[];
```

```
int         *index, *flag;
MPI_Status  *status;
```

Location

src/pt2pt/testany.c

MPI_Testsome **MPI_Testsome**

MPI_Testsome — Tests for some given communications to complete

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)

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.

Synopsis

```
int MPI_Testsome( incount, array_of_requests, outcount, array_of_indices,
    array_of_statuses )
int         incount, *outcount, array_of_indices[];
MPI_Request array_of_requests[];
MPI_Status  array_of_statuses[];
```

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

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.

Synopsis

```
int MPI_Topo_test ( comm, top_type )
MPI_Comm  comm;
int      *top_type;
```

Location

`src/topol/topo_test.c`

MPI_Type_commit **MPI_Type_commit**

MPI_Type_commit — Commits the 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.

Synopsis

```
int MPI_Type_commit ( datatype )
MPI_Datatype *datatype;
```

Location

`src/pt2pt/type_commit.c`

MPI_Type_contiguous **MPI_Type_contiguous**

MPI_Type_contiguous — Creates a contiguous datatype

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.

Synopsis

```
int MPI_Type_contiguous( count, old_type, newtype )
int          count;
MPI_Datatype old_type;
MPI_Datatype *newtype;
```

Location

`src/pt2pt/type_contig.c`

MPI_Type_extent **MPI_Type_extent**

MPI_Type_extent — Returns the extent of a datatype

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.

Synopsis

```
int MPI_Type_extent( datatype, extent )
MPI_Datatype  datatype;
MPI_Aint     *extent;
```

Location

```
src/pt2pt/type_extent.c
```

MPI_Type_free **MPI_Type_free**

MPI_Type_free — Frees the datatype

Input Parameter

datatype datatype that is freed (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.

Synopsis

```
int MPI_Type_free ( datatype )
MPI_Datatype *datatype;
```

Location

```
src/pt2pt/type_free.c
```

MPI_Type_hindexed **MPI_Type_hindexed**

MPI_Type_hindexed — Creates an indexed datatype with offsets in bytes

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.

Synopsis

```
int MPI_Type_hindexed( count, blocklens, indices, old_type, newtype )
int           count;
int           blocklens[];
MPI_Aint      indices[];
MPI_Datatype  old_type;
MPI_Datatype *newtype;
```

Location

`src/pt2pt/type_hind.c`

MPI_Type_hvector **MPI_Type_hvector**

MPI_Type_hvector — Creates a vector (strided) datatype with offset in bytes

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.

Synopsis

```
int MPI_Type_hvector( count, blocklen, stride, old_type, newtype )
int           count;
int       blocklen;
MPI_Aint      stride;
MPI_Datatype old_type;
MPI_Datatype *newtype;
```

Location

src/pt2pt/type_hvec.c

MPI_Type_indexed **MPI_Type_indexed**

MPI_Type_indexed — Creates an indexed datatype

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
10       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)
```

Synopsis

```
int MPI_Type_indexed( count, blocklens, indices, old_type, newtype )
int           count;
int        blocklens[];
int        indices[];
MPI_Datatype  old_type;
MPI_Datatype *newtype;
```

Location

src/pt2pt/type_ind.c

MPI_Type_lb **MPI_Type_lb**

MPI_Type_lb — Returns the lower-bound of a datatype

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.

Synopsis

```
int MPI_Type_lb ( datatype, displacement )
MPI_Datatype  datatype;
MPI_Aint      *displacement;
```

Location

src/pt2pt/type_lb.c

MPI_Type_size **MPI_Type_size**

MPI_Type_size — Return the number of bytes occupied by entries in the datatype

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.

Synopsis

```
int MPI_Type_size ( datatype, size )
MPI_Datatype  datatype;
int           *size;
```

Location

`src/pt2pt/type_size.c`

MPI_Type_struct **MPI_Type_struct**

MPI_Type_struct — Creates a struct datatype

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, a clarification to the standard made epsilon zero. Thus, 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.

Synopsis

```
int MPI_Type_struct( count, blocklens, indices, old_types, newtype )
int           count;
int        blocklens[];
MPI_Aint      indices[];
MPI_Datatype  old_types[];
MPI_Datatype *newtype;
```

Location

`src/pt2pt/type_struct.c`

MPI_Type_ub **MPI_Type_ub**

MPI_Type_ub — Returns the upper bound of a datatype

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.

Synopsis

```
int MPI_Type_ub ( datatype, displacement )
MPI_Datatype  datatype;
MPI_Aint      *displacement;
```

Location

src/pt2pt/type_ub.c

MPI_Type_vector **MPI_Type_vector**

MPI_Type_vector — Creates a vector (strided) datatype

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.

Synopsis

```
int MPI_Type_vector( count, blocklen, stride, old_type, newtype )
int          count;
int       blocklen;
int       stride;
MPI_Datatype old_type;
MPI_Datatype *newtype;
```

Location

src/pt2pt/type_vec.c

MPI_Unpack **MPI_Unpack**

MPI_Unpack — Unpack a datatype into contiguous memory

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.

See Also

MPI_Pack, MPI_Pack_size

Synopsis

```
int MPI_Unpack ( inbuf, insize, position, outbuf, outcount, type, comm )
void         *inbuf;
int           insize;
int          *position;
void         *outbuf;
int           outcount;
MPI_Datatype  type;
MPI_Comm      comm;
```

Location

`src/pt2pt/unpack.c`

MPI_Wait **MPI_Wait**

MPI_Wait - Waits for an MPI send or receive to complete

Input Parameter

request request (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.

Synopsis

```
int MPI_Wait ( request, status )
MPI_Request  *request;
MPI_Status   *status;
```

Location

`src/pt2pt/wait.c`

MPI_Waitall **MPI_Waitall**

MPI_Waitall — Waits for all given communications to complete

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)

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.

Synopsis

```
int MPI_Waitall(count, array_of_requests, array_of_statuses )
int         count;
MPI_Request array_of_requests[];
MPI_Status  array_of_statuses[];
```

Location

`src/pt2pt/waitall.c`

MPI_Waitany **MPI_Waitany**

MPI_Waitany — Waits for any specified send or receive to complete

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)

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.

Synopsis

```
int MPI_Waitany(count, array_of_requests, index, status )
int         count;
MPI_Request array_of_requests[];
int         *index;
MPI_Status  *status;
```

Location

`src/pt2pt/waitany.c`

MPI_Waitsome **MPI_Waitsome**

MPI_Waitsome — Waits for some given communications to complete

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)

Notes

The array of indicies are in the range `0` to `incount - 1` for C and in the range `1` to `incount` 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.

Synopsis

```
int MPI_Waitsome( incount, array_of_requests, outcount, array_of_indices,
    array_of_statuses )
int         incount, *outcount, array_of_indices[];
MPI_Request array_of_requests[];
MPI_Status  array_of_statuses[];
```

Location

`src/pt2pt/waitsome.c`

MPI_Wtick **MPI_Wtick**

MPI_Wtick — Returns the resolution of MPI_Wtime

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.

Synopsis

```
double MPI_Wtick()
```

Location

`src/env/wtick.c`

MPI_Wtime **MPI_Wtime**

MPI_Wtime — Returns an elapsed time on the calling processor

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`.

Notes for Fortran

This is a function, declared as `DOUBLE PRECISION MPI_WTIME()` in Fortran.

Synopsis

```
double MPI_Wtime()
```

Location

`src/env/wtime.c`

3 MPE routines

MPE_Add_RGB_color **MPE_Add_RGB_color**

MPE_Add_RGB_color — Adds a color to the colormap given its RGB values

Input Parameters

graph	MPE graphics handle
red,	green, blue - color levels from 0 to 65535
mapping	index of the new color

Synopsis

```
int MPE_Add_RGB_color( graph, red, green, blue, mapping )
MPE_XGraph graph;
int red, green, blue;
MPE_Color *mapping;
```

Location

`mpe/mpe_graphics.c`

MPE_CaptureFile **MPE_CaptureFile**

MPE_CaptureFile — Sets the base filename used to capture output from updates

Input Parameters

handle MPE graphics handle
fname base file name (see below)
freq Frequency of updates

Notes

The output is written in xwd format to `fname%d`, where `%d` is the number of the file (starting from zero).

Synopsis

```
int MPE_CaptureFile( handle, fname, freq )
MPE_XGraph handle;
char       *fname;
int        freq;
```

Location

`mpe/mpe_graphics.c`

MPE_Close_graphics **MPE_Close_graphics**

MPE_Close_graphics — Closes a X11 graphics device

Input Parameter

handle MPE graphics handle.

Synopsis

```
int MPE_Close_graphics( handle )
MPE_XGraph *handle;
```

Location

`mpe/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

Input Parameters

comm Communicator
rank Rank in comm

Output Parameters

grank Rank in comm world

Synopsis

```
void MPE_Comm_global_rank( comm, rank, grank )
MPI_Comm comm;
int      rank, *grank;
```

Location

mpe/getgrank.c

MPE_Counter_create **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;
```

Location

mpe/mpe_counter.c

MPE_Counter_free **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/mpe_counter.c

MPE_Counter_nxtval **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/mpe_counter.c

MPE_Decomp1d **MPE_Decomp1d**

MPE_Decomp1d — Compute a balanced decomposition of a 1-D array

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.

Synopsis

```
int MPE_Decomp1d( n, size, rank, s, e )
int n, size, rank, *s, *e;
```

Location

mpe/decomp.c

MPE_Describe_event **MPE_Describe_event**

MPE_Describe_event — Create log record describing an event type

Synopsis

```
int MPE_Describe_event( event, name )
int event;
char *name;
```

Location

mpe/mpe_log.c

MPE_Describe_state **MPE_Describe_state**

MPE_Describe_state — Create log record describing a state

Notes

Adds string containing a state def to the logfile. The format of the def is (LOG_STATE_DEF) 0 sevent eevent 0 0 "color" "name".

Synopsis

```
int MPE_Describe_state( start, end, name, color )
int start, end;
char *name, *color;
```

Location

mpe/mpe_log.c

MPE_Draw_circle **MPE_Draw_circle**

MPE_Draw_circle — Draws a circle

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

Synopsis

```
int MPE_Draw_circle( graph, centerx, centery, radius, color )
MPE_XGraph graph;
int centerx, centery, radius;
MPE_Color color;
```

Location

mpe/mpe_graphics.c

MPE_Draw_line **MPE_Draw_line**

MPE_Draw_line — Draws a line on an X11 display

Input Parameters

handle	MPE graphics handle
x1,y1	pixel position of one end of the line to draw. Coordinates are upper-left origin (standard X11)
x2,y2	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.

Synopsis

```
int MPE_Draw_line( handle, x1, y1, x2, y2, color )
MPE_XGraph handle;
int        x1, y1, x2, y2;
MPE_Color  color;
```

Location

mpe/mpe_graphics.c

MPE_Draw_logic **MPE_Draw_logic**

MPE_Draw_logic — Sets logical operation for laying down new pixels

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

Synopsis

```
int MPE_Draw_logic( graph, function )
MPE_XGraph graph;
int function;
```

Location

mpe/mpe_graphics.c

MPE_Draw_point **MPE_Draw_point**

MPE_Draw_point — Draws a point on an X Windows display

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.

Synopsis

```
int MPE_Draw_point( handle, x, y, color )
MPE_XGraph handle;
int        x, y;
MPE_Color  color;
```

Location

mpe/mpe_graphics.c

MPE_Draw_points **MPE_Draw_points**

MPE_Draw_points — Draws points on an X Windows display

Input Parameters

handle	MPE graphics handle
points	list of points to draw
npoints	number of points to draw

Synopsis

```
int MPE_Draw_points( handle, points, npoints )
MPE_XGraph handle;
MPE_Point *points;
int npoints;
```

Location

mpe/mpe_graphics.c

MPE_Draw_string **MPE_Draw_string**

MPE_Draw_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
string	text string to be drawn
color	color of the text

Synopsis

```
int MPE_Draw_string( graph, x, y, color, string )
MPE_XGraph graph;
int x, y;
MPE_Color color;
char *string;
```

Location

mpe/mpe_graphics.c

MPE_Errors_call_debugger **MPE_Errors_call_debugger**

MPE_Errors_call_debugger — On an error, print a message and (attempt) to start the specified debugger on the program

Input Parameters

pgm	Name of the program.
dbg	Name of the debugger. If null, use a default (usually dbx)
args	arguments to use in generating the debugger. This allows things like "`xterm -e dbx pgm pid`", or "`xdbx -geometry +%d+%d pgm pid`". The list should be null terminated. (The `%d %d` format is not yet supported).

Notes

You may need to ignore some signals, depending on the signals that the MPICH and underlying communications code is using. You can do this in dbx by adding "ignore signal-name" to your .dbxinit file. For example, to ignore `SIGUSR1`, use "`ignore USR1`".

Currently, there is no Fortran interface for this routine.

Synopsis

```
void MPE_Errors_call_debugger( pgm, dbg, args )
char *pgm, *dbg, **args;
```

Location

`mpe/dbxerr.c`

MPE_Fill_circle **MPE_Fill_circle**

MPE_Fill_circle — Fills a circle

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

Synopsis

```
int MPE_Fill_circle( graph, centerx, centery, radius, color )
MPE_XGraph graph;
int centerx, centery, radius;
MPE_Color color;
```

Location

mpe/mpe_graphics.c

MPE_Fill_rectangle **MPE_Fill_rectangle**

MPE_Fill_rectangle — Draws a filled rectangle on an X11 display

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.

Synopsis

```
int MPE_Fill_rectangle( handle, x, y, w, h, color )
MPE_XGraph handle;
int        x, y, w, h;
MPE_Color  color;
```

Location

mpe/mpe_graphics.c

MPE_Finish_log **MPE_Finish_log**

MPE_Finish_log — Send log to master, who writes it out

Notes

This routine dumps a logfile in alog format

Synopsis

```
int MPE_Finish_log( filename )
char *filename;
```

Location

mpe/mpe_log.c

MPE_GetTags **MPE_GetTags**

MPE_GetTags — Returns tags that can be used in communication with a communicator

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 guarenteed to be unique within `comm_out`.

See Also

MPE_ReturnTags

Synopsis

```
int MPE_GetTags( comm_in, ntags, comm_out, first_tag )
MPI_Comm comm_in, *comm_out;
int      ntags, *first_tag;
```

Location

`mpe/privtags.c`

MPE_Get_mouse_press **MPE_Get_mouse_press**

MPE_Get_mouse_press — Waits for mouse button press

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.

Synopsis

```
MPE_Get_mouse_press( graph, x, y, button )
MPE_XGraph graph;
int *x, *y, *button;
```

Location

`mpe/xmouse.c`

MPE_IO_Stdout_to_file **MPE_IO_Stdout_to_file**

MPE_IO_Stdout_to_file — Re-direct stdout to a file

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.

Synopsis

```
void MPE_IO_Stdout_to_file( name, mode )
char *name;
int  mode;
```

Location

`mpe/mpe_io.c`

MPE_Iget_mouse_press **MPE_Iget_mouse_press**

MPE_Iget_mouse_press — Checks for mouse button press 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.

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

Synopsis

```
int MPE_Iget_mouse_press( graph, x, y, button, wasPressed )
MPE_XGraph graph;
int *x, *y, *button, *wasPressed;
```

Location

`mpe/xmouse.c`

MPE_Initlog **MPE_Initlog**

MPE_Initlog — Initialize for logging

Synopsis

```
int MPE_Init_log()
```

Location

`mpe/mpe_log.c`

MPE_Line_thickness **MPE_Line_thickness**

MPE_Line_thickness — Sets thickness of lines

Input Parameters

graph MPE graphics handle
thickness integer specifying how many pixels wide lines should be

Synopsis

```
int MPE_Line_thickness( graph, thickness )
MPE_XGraph graph;
int thickness;
```

Location

mpe/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;
```

Location

mpe/mpe_log.c

MPE_Log_message **MPE_Log_message**

MPE_Log_message — log the sending or receiving of a message

Synopsis

```
static int MPE_Log_message( type, otherParty, tag, size )
int type, otherParty, tag, size;
```

Location

mpe/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/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/mpe_log.c

MPE_Make_color_array **MPE_Make_color_array**

MPE_Make_color_array — Makes an array of color indices

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.

Synopsis

```
int MPE_Make_color_array( handle, ncolors, array )
MPE_XGraph handle;
int        ncolors;
MPE_Color  array[];
```

Location

mpe/mpe_graphics.c

MPE_Num_colors **MPE_Num_colors**

MPE_Num_colors — Gets the number of available colors

Input Parameter

handle MPE graphics handle

Output Parameter

nc Number of colors available on the display.

Synopsis

```
int MPE_Num_colors( handle, nc )
MPE_XGraph handle;
int        *nc;
```

Location

mpe/mpe_graphics.c

MPE_Open_graphics **MPE_Open_graphics**

MPE_Open_graphics — (collectively) opens an X Windows display

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 ":", 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).

Synopsis

```
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+4];
int        x, y;
int        w, h;
int        is_collective;
```

Location

mpe/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

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.

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;
```

Location

```
mpe/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

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.

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;
```

Location

mpe/examine.c

MPE_Ptime **MPE_Ptime**

MPE_Ptime — Returns process time stamp

Notes

This routine returns the CPU time charged to the process only if the PETSC package is available (it uses the PETSc routine `SYGetCPUTime`). Otherwise, it returns `MPI_Wtime` (wall-clock time).

Synopsis

```
double MPE_Ptime()
```

Location

mpe/timers.c

MPE_ReturnTags **MPE_ReturnTags**

MPE_ReturnTags — Returns tags allocated with MPE_GetTags.

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

Synopsis

```
int MPE_ReturnTags( comm, first_tag, ntags )
MPI_Comm comm;
int      first_tag, ntags;
```

Location

mpe/privtags.c

MPE_Seq_begin **MPE_Seq_begin**

MPE_Seq_begin — Begins a sequential section of code.

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 controling terminal (in other words, even if you flush the output, you may not get the data in the order that you want).

Synopsis

```
void MPE_Seq_begin( comm, ng )
MPI_Comm comm;
int      ng;
```

Location

mpe/mpe_seq.c

MPE_Seq_end **MPE_Seq_end**

MPE_Seq_end — Ends a sequential section of code.

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.

Synopsis

```
void MPE_Seq_end( comm, ng )
MPI_Comm comm;
int      ng;
```

Location

mpe/mpe_seq.c

MPE_Signals_call_debugger **MPE_Signals_call_debugger**

MPE_Signals_call_debugger — Process-killing signals invoke the MPE error handler

Notes

This invokes the MPE error handler which prints some information about the signal and then attempts to start the debugger. Some systems will not support calling the debugger.

You may need to ignore some signals, depending on the signals that the MPICH and underlying communications code is using. You can do this in dbx by adding "`ignore signal-name`" to your .dbxinit file. For example, to ignore `SIGUSR1`, use "`ignore USR1`".

Currently, there is no Fortran interface for this routine.

Synopsis

```
void MPE_Signals_call_debugger()
```

Location

mpe/dbxerr.c

MPE_Start_log **MPE_Start_log**

MPE_Start_log — Begin logging of events

Synopsis

```
int MPE_Start_log()
```

Location

mpe/mpe_log.c

MPE_Stop_log **MPE_Stop_log**

MPE_Stop_log — Stop logging events

Synopsis

```
int MPE_Stop_log()
```

Location

mpe/mpe_log.c

MPE_TagsEnd **MPE_TagsEnd**

MPE_TagsEnd — Returns the private keyval.

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`.

Synopsis

```
int MPE_TagsEnd()
```

Location

mpe/privtags.c

MPE_Update **MPE_Update**

MPE_Update — Updates an X11 display

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.

Synopsis

```
int MPE_Update( handle )
MPE_XGraph handle;
```

Location

mpe/mpe_graphics.c

MPE_Wtime **MPE_Wtime**

MPE_Wtime — Returns wall clock time stamp

Synopsis

```
double MPE_Wtime()
```

Location

```
mpe/timers.c
```

4 ADI routines

ADI **ADI**

ADI Parameters - Compile-time parameters for the ADI

The ADI makes use of a number of compile-time parameters both to decide what to do and to indicate to the MPICH code what capabilities it has. In addition, specific implementations of the ADI also make use of compile-time parameters. They are all documented here.

Parameters for the ADI

These parmeters apply to any ADI, independant of implementation.

MPID_CAN_SEND_CONTIG
: Obsolete; ignored

MPID_HAS_HETERO
: If set, environment is heterogeneous

MPID_DEVICE_CODE
: If set, indicates compiling the ADI itself. Should NOT be set when compiling the routines in mpich/src/* .

MPID_NEEDS_WORLD_SRC_INDICES
: If set, indicates that the Probe routines need "rank in world" instead of "rank in communicator". This is used in mpid/meiko .

MPID_PACK_IN_ADVANCE
: If set, asks the MPI routines to give the ADI only contiguous data. Currently, all devices set this.

MPID_RETURN_PACKED
: Receive counterpart of `MPID_PACK_IN_ADVANCE`

Parameters for the Chameleon version of the ADI

These parameters apply to the version of the ADI in mpid/chameleon, and to the ones derived from it (mpid/ch_*).

MPID_DEBUG_ALL
: If set, include code (switchable at runtime) for debugging

MPID_HAS_SHARED_MEM
: If set, environment has shared memory. Intended only for collective routines

MPID_H_INC
: Used to protect include file if included twice

MPID_HAS_DEBUG
: If set, ADI has some debugging features

MPID_LIMITED_BUFFERS
: Indicates an underlying message-passing system with small buffers (like IBM MPL); causes the ADI to be more aggressive about polling for incoming messages

MPID_TEST_SYNC

For debugging only (of synchronous messages)

MPID_TINY_BUFFERS
If set, underlying message passing system has essentially no buffering (this is more strict that `MPID_LIMITED_BUFFERS`)

MPID_USE_GET
If set, use shared memory operations

MPID_USE_SEND_BLOCK
If set, use a special send routine that may block when possible. Used in mpid/ch_cmmd to conserve tports.

Parameters for the channel version of the ADI

The following all refer to the "channel" interface and control details of the message packets that are sent.

MPID_CACHE_ALIGN
Used for shared memory; requests that shared memory locks be cached aligned

MPID_PKT_DEF
Protects include file

MPID_PKT_DYNAMIC_RECV,MPID_PKT_DYNAMIC_SEND
These two indicate whether send/receive routines take address of packet or packet. Used only in the shared memory version

MPID_PKT_GALLOC
If set, provides a global allocation of a packet

MPID_PKT_GET_NEEDS_ACK
If set, the shared-memory "get" operation requires an acknowledgement (for example, if a single address space is used)

MPID_ADI_MUST_SENDSELF
If set, the ADI handles the case of sending a message from one process to the same process with special code, bypassing the channel interface code.

Packet Features

These provide optional declarations in a packet for the length of a packet, a link (shared memory), and the source (sending process). Used in shared memory and raw tcp. The "INCLUDE" versions define DECL/SET/GET versions.

MPID_PKT_INCLUDE_LEN
Include a packet size field (`int pkt_len:32`) in the packet definition

MPID_PKT_INCLUDE_LINK
Include a link field (`union _MPID_PKT_T *next`) in the packet definition

MPID_PKT_INCLUDE_SRC
Include a source in `MPI_COMM_WORLD` field (`int src:32`) in the packet definition.

MPID_PKT_LEN_DECL
Appears in the packet definition; contains the actual declaration (if any). For internal use only.

MPID_PKT_LINK_DECL
Appears in the packet definition; contains the actual declaration (if any). For internal use only.

MPID_PKT_SRC_DECL
Appears in the packet definition; contains the actual declaration (if any). For internal use only.

MPID_PKT_PRIVATE
Used to add "private" data at the *top* of the packet. Used in some shared memory versions.

MPID_PKT_XDR_DECL
Used to define a flag for heterogeneous support (message is using XDR)

Chameleon-MPI **Chameleon-MPI**

Chameleon-MPI transport (device) implementation

This simple Chameleon implementation is meant to work with most message passing systems and does not rely on large amounts of message buffering. Messages are sent in one or two packets, depending on the length. If the message is short (< `MPID_PKT_DATA_SIZE`), then the header and message are placed into a single message and sent together. Long messages are sent without copying the message by sending the header in a separate packet.

This Chameleon implementation depends on adequate buffering by the message-passing system. Because there is no asynchronous notification of messages in Chameleon (because most transport layers do not provide this important functionality), it is difficult to implement a packet-based system (i.e., one that divides messages into multiple packets and manages all buffering).

Comments on the implementation

There are many things that can be done to improve the performance of the implementation of the ADI on various platforms. Some of these have been added to the current implementation with ifdef"s (see the MPICH configure for more discussion of what is currently available).

Collective **Collective**

Collective — Support in the ADI for collective operations

Discussion

On some systems, there is special hardware for some collective operations. On others, there are special techniques that can be applied to significantly speed up some collective operations.

It is not necessary to support ANY collective operations within the ADI. However, the ADI does provide a way for the ADI to perform some operations. In addition, these operations may be provided for just MPI_COMM_WORLD or for any communicator.

The ADI provides for the following classes of collective operations

MPID_Barrier
: Barrier

MPID_Reduce_op_type
: Reduction (op == sum, prod, etc, type = double, int, etc). Define MPID_Reduce if ANY are defined

MPID_Scan_op_type
: Scan reduction. Define `MPID_Scan` if *any* are defined.

MPID_Bcast Broadcast (1 to all)

MPID_AlltoAll
: All to all

These routines all handle a *single element*, since this is an important and common case.

Since these may require the creation of special data structures, the ADI routine `MPID_Comm_init` must be called before these may be used. `MPID_Comm_free` must be called when a communicator is destroyed to ensure that any resources are returned.

All of these "routines" are actually macros, allowing the API code to contain code like

```
#ifdef MPID_Barrier
if (comm->AdiBarrier) MPID_Barrier( comm->AdiCtx, comm );
else
#endif
{
```

```
... code using pt-2-pt
}
```

Not yet decided: are the reduction and scan reduced to a single root or to all (I"d prefer to all).

MPID_Abort **MPID_Abort**

MPID_Abort — Terminates a parallel program

Input Parameter

code	Value to return to invocation environment as exit status
ctx	ADI context returned by `MPID_Init`

Synopsis

```
void MPID_Abort( void * ctx, int code )
```

Notes

This call terminates all processes in a parallel program.

It is not always possible to return code as the status.

MPID_Alloc_recv_handle **MPID_Alloc_recv_handle**

MPID_Alloc_recv_handle — Allocates and initializes device recv handle

Input Parameter

ctx	ADI context returned by `MPID_Init`
a	Address of device recv handle in APIs send structure

Synopsis

```
void MPID_Alloc_recv_handle( void * ctx, MPID_SHANDLE *mpid_recv_handle )
```

Notes

In systems where the device handle is statically allocated, this routine may be used to initialize the fields of that handle. In particular, it may set fields such as "completed" to false.

MPID_Alloc_send_handle **MPID_Alloc_send_handle**

MPID_Alloc_send_handle — Allocates and initializes device send handle

Input Parameter

ctx ADI context returned by `MPID_Init`
a Address of device send handle in APIs send structure

Synopsis

```
void MPID_Alloc_send_handle( void * ctx, MPID_SHANDLE *mpid_send_handle )
```

Notes

In systems where the device handle is statically allocated, this routine may be used to initialize the fields of that handle. In particular, it may set fields such as "completed" to false.

MPID_Barrier **MPID_Barrier**

MPID_Barrier — Performs a barrier synchronization

Input Parameters

ctx ADI context returned by `MPID_Init`
comm MPI Communicator

Synopsis

```
void MPID_Barrier( void *ctx, MPI_Comm comm )
```

MPID_Blocking_recv **MPID_Blocking_recv**

MPID_Blocking_recv — Performs a blocking receive

Input Parameter

ctx ADI context returned by `MPID_Init`
dmpi_recv_handle
APIs receive handle

Synopsis

```
int MPID_Blocking_recv( void * ctx, MPIR_RHANDLE *dmpi_recv_handle )
```

MPID_Blocking_send **MPID_Blocking_send**

MPID_Blocking_send — Performs a blocking send.

Input Parameter

ctx ADI context returned by `MPID_Init`
dmpi_send_handle
APIs send handle

Synopsis

```
int MPID_Blocking_send( void * ctx, MPIR_SHANDLE *dmpi_send_handle )
```

MPID_Blocking_send_ready **MPID_Blocking_send_ready**

MPID_Blocking_send_ready — Performs a ready blocking send

Input Parameter

ctx ADI context returned by `MPID_Init`
dmpi_send_handle
APIs send handle

Synopsis

```
int MPID_Blocking_send_ready( void * ctx, MPIR_SHANDLE *dmpi_send_handle )
```

Notes

This may be implemented as `MPID_Blocking_send`

MPID_Cancel **MPID_Cancel**

MPID_Cancel — Cancels a pending communication operation.

Input Parameter

ctx ADI context returned by MPID_Init
r pointer to an API message request

Synopsis

```
void MPID_Cancel( void * ctx, MPIR_CHANDLE *r )
```

Notes

The actual freeing of internal space must be done with the appropriate `MPID_FREE_SEND/RECV_HANDLE` calls.

This routine may have little effect. If the message has already been delivered, it has no effect.

MPID_Check_device **MPID_Check_device**

MPID_Check_device — Checks for pending operations and handles if found.

Input Parameter

blocking Indicates whether the operation is blocking or not. The values `MPID_BLOCKING` and `MPID_NOT_BLOCKING` can be used.
ctx ADI context returned by `MPID_Init`

Synopsis

```
void MPID_Check_device( void * ctx, int blocking )
```

Notes

This routine should be called by the API frequently *if* the ADI requires it. For example, if the ADI is not a separate thread and does not use signals to operate asynchronously.

Some ADI routines may also use this to process pending ADI events.

MPID_Check_incoming **MPID_Check_incoming**

MPID_Check_incoming

Input Parameter

ctx ADI context returned by `MPID_Init`
blocking Indicates whether the operation is blocking or not. The values `MPID_BLOCKING` and `MPID_NOT_BLOCKING` can be used.

Returns

-1 if nonblocking and no messages pending

Synopsis

```
int MPID_Check_incoming( void * ctx, MPID_BLOCKING_TYPE blocking )
```

Notes

This routine handles testing for a single incoming message. For people interested in active messages, this routine is essentially the handler for the incoming active messages.

MPID_Clr_completed **MPID_Clr_completed**

MPID_Clr_completed — Mark a request as not completed

Input Parameter

ctx ADI context returned by `MPID_Init`
request MPI_Request handle to set.

Notes

This is used within the Chameleon ADI implementation only

MPID_Comm_free **MPID_Comm_free**

MPID_Comm_free — Frees the ADI"s use of a communicator.

Input Parameters

ctx ADI context returned by `MPID_Init`
comm MPI Communicator

Synopsis

```
void MPID_Comm_free( void *ctx, MPI_Comm comm )
```

Notes

This routine is responsible for freeing any data areas needed to perform any ADI-supported collective operations (such as `MPID_Barrier`).

MPID_Comm_init **MPID_Comm_init**

MPID_Comm_init — Initializes the ADI"s use of a communicator

Input Parameters

ctx ADI context returned by `MPID_Init`
parent MPI Communicator of parent
comm MPI Communicator

Synopsis

```
void MPID_Comm_init( void *ctx, MPI_Comm comm )
```

Notes

This routine is responsible for initializing any data areas needed to perform any ADI-supported collective operations (such as `MPID_Barrier`). In addition, this routine is collective in the parent communicator. If the parent is `NULL`, comm must be `MPI_COMM_WORLD` and all processes must call this routine.

MPID_Complete_recv **MPID_Complete_recv**

MPID_Complete_recv — Completes a receive of a message.

Input Parameter

ctx ADI context returned by `MPID_Init`

dmpi_recv_handle
APIs receive handle

Synopsis

```
int MPID_Complete_recv( void * ctx, MPIR_RHANDLE *dmpi_recv_handle )
```

Notes

This routine completes a send begun with `MPID_Post_recv`.

MPID_Complete_send **MPID_Complete_send**

MPID_Complete_send — Completes a send of a message.

Input Parameter

ctx ADI context returned by MPID_Init

dmpi_send_handle
APIs send handle

Synopsis

```
int MPID_Complete_send( void * ctx, MPIR_SHANDLE *dmpi_send_handle )
```

Notes

This routine completes a send begun with `MPID_Post_send`, `MPID_Post_send_ready`, or `MPID_Post_send_sync`.

MPID_Dest_byte_order **MPID_Dest_byte_order**

MPID_Dest_byte_order — Returns the byte ordering desired by the destination

Input Parameter

dest Absolute rank or destination

Returns

One of `MPID_H_NONE`, `MPID_H_LSB`, `MPID_H_MSB`, `MPID_H_XDR`

Synopsis

```
int MPID_Dest_byte_order( int dest )
```

MPID_End **MPID_End**

MPID_End — Ends the operation of the ADI

Input Parameter

ctx ADI context returned by `MPID_Init`

Synopsis

```
void MPID_End( void * ctx )
```

Notes

This routine must be called at the end of the parallel part of a program. Normally, this is just before the program exits. Once `MPID_End` is called, no other MPID routines, including `MPID_Init`, may be called. Note that by the time this routine is called, much of the MPI environment may already have been dismantled. For example, the communicators and datatypes may have been freed before this routine is called.

If the macro `MPID_END_NEEDS_BARRIER` is defined, the code that calls `MPID_End` must call `MPI_BARRIER( MPI_COMM_WORLD )` before calling `MPID_End`. This helps

---

MPID_Flags **MPID_Flags**

MPID_Flags — Describes the run-time data collection and debugging flags

Notes

The MPID/ADI implementation collects a number of statistics and provides a variety of debugging options.

This interface is currently under development

MPID_Free_recv_handle **MPID_Free_recv_handle**

MPID_Free_recv_handle — Frees device recv handle

Input Parameter

ctx ADI context returned by `MPID_Init`
a Address of device recv handle in APIs recv structure

Synopsis

```
void MPID_Free_recv_handle( void * ctx, MPID_RHANDLE *mpid_recv_handle )
```

Notes

In systems where the device handle is statically allocated, this routine may be used to free any areas that may have been created by the recv operation.

MPID_Free_send_handle **MPID_Free_send_handle**

MPID_Free_send_handle — Frees device send handle

Input Parameter

ctx	ADI context returned by `MPID_Init`
a	Address of device send handle in APIs send structure

Synopsis

```
void MPID_Free_send_handle( void * ctx, MPID_SHANDLE *mpid_send_handle )
```

Notes

In systems where the device handle is statically allocated, this routine may be used to free any areas that may have been created by the send operation.

MPID_HAS_HETERO **MPID_HAS_HETERO**

MPID_HAS_HETERO — Indicates that heterogeneous processors may be present

Notes

This is a macro that is defined when the ADI supports heterogeneous processing elements. APIs that are to operate in a heterogeneous environment need to ensure that message are properly prepared. See also the discussion on heterogeneous support, `MPID_IS_HETERO`

MPID_IS_HETERO **MPID_IS_HETERO**

MPID_IS_HETERO — Indicates that heterogeneous processors are present

Notes

This is a global variable that can be used to determine if an ADI that could contain heterogeneous processors actually does. For example, if the ADI can connect disimilar workstations but the active collection contains only similar workstations, then `MPID_HAS_HETERO` would be defined, but `MPID_IS_HETERO` would be false.

See also the discussion on heterogeneous support.

MPID_Init **MPID_Init**

MPID_Init — Initializes the ADI

Input Parameters

argc	Pointer to an integer containing the number of arguments
argv	Pointer to the argument vector

Returns

Pointer that may be passed back to these routines (context).

Synopsis

```
void *MPID_Init( int *argc, char ***argv )
```

Notes

If called from a C program with main(argc,argv), the correct calling sequence is `MPID_Init(&argc,&argv)`.

This routine may (or may not) also initialize the parallel processes themselves. That is, when this routine is called, it may create the requested number of parallel processes, or it may simply make use of an already existing set. What the case is, and how the number of processes is specified, is specific to any particular implementation and environment.

As an example, the Chameleon version of the ADI has these properties

```
Transport Layer             -   MPID_Init starts processes
-------------------------------------------------------------
p4, pvm, EUIH               -   yes
Paragon                     -   depends on Paragon
Intel i860, Delta, CM5, POE -   no
```

The pointer returned by this routine may be null; it is provided to allow for multiple, simultaneous ADIs. For example, if the pointer returned contains a structure with pointers to the ADI routines, and the macros for the operations choose the actual function from this structure, then a single user-application can easily switch between ADIs, depending, for example, on the MPI Communicator in use.

MPID_Init_communicator **MPID_Init_communicator**

MPID_Init_communicator — Initializes a new communictors

Input Parameters

ctx	ADI context returned by `MPID_Init`
comm	MPI Communicator

Synopsis

```
int MPID_Init_communicator( void * ctx, MPI_Comm comm )
```

Notes

Not yet used.

This interface is intended to let the ADI optimize for various collective operations by detecting whether special features are available. Note that `MPI_Pack` is a collective operation in the sense that the result must be valid for all members of a communicator; by letting the ADI initialize a communicator, it can pre-determine if, for example, all members use the same data-representation.

For future extensions for collective operations and shared-memory systems, this entry could allocate data-structures for performing various operations.

MPID_Iprobe **MPID_Iprobe**

MPID_Iprobe — Returns whether a message of the specified type is available

Input Parameters

ctx	ADI context returned by `MPID_Init`
tag	Message tag
source	Message source (sending processor). Measured in terms of rank of the communicator
context_id	Context id value of communicator.

Output Parameters

found	Set to true if a message is available, false otherwise
status	MPI status object; has values set if found is true

Synopsis

```
int MPID_Iprobe( void * ctx, int tag, int source, int context_id,
int *found, MPI_Status *status )
```

Notes

Since this routine does not have a communicator argument, there is no easy way to convert the `source` argument from the value here (local to the communicator that `context_id` belongs to) and to the source relative to `MPI_COMM_WORLD`. To force `source` to be relative to `MPI_COMM_WORLD` instead of the communicator, the ADI should define `MPID_NEEDS_WORLD_SRC_INDICES` in `dm.h`.

MPID_Message_protocol **MPID_Message_protocol**

MPID_Message_protocol — Describes the message prototcol used by the Chameleon ADI

Notes

All messages are sent with a short header message. In addition, long messages may be sent separately, in-order to make best use of vendor-provided message-passing interfaces.

In the receive code, there are routines to extract the data from short and long messages (`MPID_Copy_body_short`, `MPID_Copy_body_long`).

In the send code, the routines are `MPID_post_send_short` and `MPID_post_send_long`. `MPID_post_send_long` may use nonblocking operations.

Notes

A future version might provide nonblocking versions of the extract long message version (`MPID_nextract_long`, `MPID_wextract_long`) instead of simply `MPID_Copy_body_long`.

MPID_Myrank **MPID_Myrank**

MPID_Myrank — Provides the rank of the calling process

Input Parameter

ctx ADI context returned by `MPID_Init`

Output Parameter

rank rank, in the range 0 to `Mysize-1` (see `MPID_Mysize`) of the calling process among all of the processes.

Synopsis

```
void MPID_Myrank( void * ctx, int *rank )
```

MPID_Mysize **MPID_Mysize**

MPID_Mysize — Provides the number of processes

Input Parameter

ctx ADI context returned by `MPID_Init`

Output Parameter

size number of processes in the set of processes.

Synopsis

```
void MPID_Mysize( void * ctx, int *size )
```

MPID_Node_name **MPID_Node_name**

MPID_Node_name — Provides the name of the processor.

Input Parameter

ctx ADI context returned by `MPID_Init`
len Length of "name" argument

Output Parameter

name Name of the processor on which this call took place. It should identify the specific hardware when possible

Synopsis

```
void MPID_Node_name( void * ctx, char *name, int len )
```

MPID_PACK_IN_ADVANCE **MPID_PACK_IN_ADVANCE**

MPID_PACK_IN_ADVANCE — Indicates that the ADI prefers to be given a single buffer

Notes

This is a macro that indicates that the data will be provided in the data-area specified by the "start" field of the device handle. If the message is not contiguous, the API is responsible for packing the message (it may use `MPIR_Pack`).

More advanced ADI implementations may inform the calling routines (the API code) that the ADI can handle more complicated data, such as `MPID_DOES_STRIDED`. The exact interface for this has not been determined.

MPID_Post_send **MPID_Post_send**

MPID_Post_send — Starts a send of a message

Input Parameter

ctx ADI context returned by `MPID_Init`
dmpi_recv_handle
APIs send handle

Synopsis

```
int MPID_Post_send( void * ctx, MPIR_SHANDLE *dmpi_send_handle )
```

Notes

This routine only begins the send. `MPID_Complete_send` must be called to finish the send, even if it was a blocking operation

---

MPID_Post_send_ready **MPID_Post_send_ready**

MPID_Post_send_ready — Starts a send of a message with ready mode

Input Parameter

ctx ADI context returned by `MPID_Init`
dmpi_recv_handle
APIs send handle

Synopsis

```
int MPID_Post_send_ready( void * ctx, MPIR_SHANDLE *dmpi_send_handle )
```

Notes

This routine only begins the send. `MPID_Complete_send` must be called to finish the send, even if it was a blocking operation. The matching receive should already be posted.

It is legal to implement this with `MPID_Post_send`.

MPID_Post_send_sync **MPID_Post_send_sync**

MPID_Post_send_sync — Starts a send of a message with synchronous mode

Input Parameter

ctx ADI context returned by `MPID_Init`
dmpi_recv_handle
APIs send handle

Synopsis

```
int MPID_Post_send_sync( void * ctx, MPIR_SHANDLE *dmpi_send_handle )
```

Notes

This routine only begins the send. `MPID_Complete_send` must be called to finish the send, even if it was a blocking operation. This send will not complete until the destination *begins* the matching receive.

MPID_Probe **MPID_Probe**

MPID_Probe — Returns status for a matching message; waits if message is not available.

Input Parameters

ctx	ADI context returned by `MPID_Init`
tag	Message tag
source	Message source (sending processor). Measured in terms of rank of the communicator
context_id	Context id value of communicator.

Output Parameters

status	MPI status object; has values set if found is true

Synopsis

```
int MPID_Probe( void * ctx, int tag, int source, int context_id,
MPI_Status *status )
```

Notes

Since this routine does not have a communicator argument, there is no easy way to convert the `source` argument from the value here (local to the communicator that `context_id` belongs to) and to the source relative to `MPI_COMM_WORLD`. To force `source` to be relative to `MPI_COMM_WORLD` instead of the communicator, the ADI should define `MPID_NEEDS_WORLD_SRC_INDICES` in `dm.h`.

MPID_Procinfo **MPID_Procinfo**

MPID_Procinfo — Data structure that contains information on the processes

Notes

This structure is undergoing development. See `MPID_Dest_byte_order`.

MPID_RETURN_PACKED **MPID_RETURN_PACKED**

MPID_RETURN_PACKED — Indicates that the ADI prefers to return a single buffer

Notes

This is a macro that indicates that the data will be returned in the data-area specified by the "start" field of the device handle. If the message is not contiguous, the API is responsible for unpacking the message (with the routines MPIR_SetupUnPackMessage and MPIR_UnPackMessage)

MPID_Reduce_sum_double **MPID_Reduce_sum_double**

MPID_Reduce_sum_double — Reduce a single double value by sum on all processes

Input Parameters

ctx ADI context returned by `MPID_Init`
sendbuf Contribution from each process
comm MPI Communicator

Output Parameter

recvbuf Contains (on ALL processes) the output value

Synopsis

```
void MPID_Reduce_sum_double( void *ctx, double *sendbuf, double *recvbuf,
MPI_Comm comm )
```

MPID_Reduce_sum_int **MPID_Reduce_sum_int**

MPID_Reduce_sum_int — Reduce a single int value by sum on all processes

Input Parameters

ctx ADI context returned by `MPID_Init`
sendbuf Contribution from each process
comm MPI Communicator

Output Parameter

recvbuf Contains (on ALL processes) the output value

Synopsis

```
void MPID_Reduce_sum_int( void *ctx, int *sendbuf, int *recvbuf,
MPI_Comm comm )
```

MPID_Reuse_recv_handle **MPID_Reuse_recv_handle**

MPID_Reuse_recv_handle — Reinitializes device recv handle

Input Parameter

ctx ADI context returned by `MPID_Init`
a Address of device recv handle in APIs send structure

Synopsis

```
void MPID_Reuse_recv_handle( void * ctx, MPID_RHANDLE *mpid_recv_handle )
```

Notes

In systems where the device handle is statically allocated, this routine may be used to reset fields such as "completed" to false. This routine is used for persistent message-passing requests

---

MPID_Reuse_send_handle **MPID_Reuse_send_handle**

MPID_Reuse_send_handle — Reinitializes device send handle

Input Parameter

ctx ADI context returned by `MPID_Init`
a Address of device send handle in APIs send structure

Synopsis

```
void MPID_Reuse_send_handle( void * ctx, MPID_SHANDLE *mpid_send_handle )
```

Notes

In systems where the device handle is statically allocated, this routine may be used to reset fields such as "completed" to false. This routine is used for persistent message-passing requests

---

MPID_Set_completed **MPID_Set_completed**

MPID_Set_completed — Mark a request as completed

Input Parameter

ctx ADI context returned by `MPID_Init`
request MPI_Request handle to set.

Notes

This is used within the Chameleon ADI implementation only

MPID_Test_recv **MPID_Test_recv**

MPID_Test_recv — Tests for the completion of a receive request.

Input Parameter

ctx ADI context returned by `MPID_Init`
dmpi_recv_handle
APIs recv handle

Return value

True (!= 0) if the request has completed, False (0) otherwise.

Notes

This routine must return true if the request either has completed or would complete if `MPID_Check_device` were called. In other words, it must be possible to have a loop

```
while (!MPID_Test_recv( ctx, request )) ;
```

that will terminate when the message arrives, without requiring an explicit call to another ADI routine.

MPID_Test_request **MPID_Test_request**

MPID_Test_request — Tests a request for completion.

Input Parameter

ctx ADI context returned by `MPID_Init`
request MPI_Request handle to test.

Return value

True (!= 0) if the request has completed, False (0) otherwise.

Notes

This routine must return true if the request either has completed or would complete if `MPID_Check_device` were called. In other words, it must be possible to have a loop

```
while (!MPID_Test_request( ctx, request )) ;
```

that will terminate when the message has been sent or received, without requiring an explicit call to another ADI routine.

MPID_Test_send **MPID_Test_send**

MPID_Test_send — Tests for the completion of a send request.

Input Parameter

ctx ADI context returned by `MPID_Init`
dmpi_send_handle
APIs send handle

Return value

True (!= 0) if the request has completed, False (0) otherwise.

Notes

This routine must return true if the request either has completed or would complete if `MPID_Check_device` were called. In other words, it must be possible to have a loop

```
while (!MPID_Test_send( ctx, request )) ;
```

that will terminate when the message has been sent, without requiring an explicit call to another ADI routine.

MPID_Version_name **MPID_Version_name**

MPID_Version_name — Provides a text string describing the version of the ADI

Input Parameter

ctx ADI context returned by `MPID_Init`

Output Parameter

name Name of the version of the ADI. Must be a string of at least 128 characters

Synopsis

```
void MPID_Version_name( void * ctx, char *name )
```

MPID_Wtick **MPID_Wtick**

MPID_Wtick — Returns the resolution of the timer `MPID_Wtime`

Input Parameter

ctx ADI context returned by `MPID_Init`

Synopsis

```
double MPID_Wtick( void * ctx )
```

MPID_Wtime **MPID_Wtime**

MPID_Wtime — Returns the time, relative to some arbitrary point.

Input Parameter

ctx ADI context returned by `MPID_Init`

Synopsis

```
double MPID_Wtime( void * ctx )
```

Notes

This routine may return a different value on eacy processor even when called at the same "time". The only quantity that is meaningful is the difference between two results of `MPID_Wtime`. The resolution of this time is provide by `MPID_Wtick`; while `MPID_Wtime` should provide a high-resolution time, this is not required or guarenteed.

The time measured is elapsed or wall-clock time, not CPU time.

This may also be provided as a macro in the file `dm.h`.

MPID_Wtime_is_global **MPID_Wtime_is_global**

MPID_Wtime_is_global — Returns the resolution of the timer `MPID_Wtime`

Return value

True (1) if the value from `MPID_Wtime` is synchronized on all processes, False (0) otherwise.

Synopsis

```
int MPID_Wtime_is_global( void )
```

Notes

This is often implemented with a definition

```
#define MPID_Wtime_is_global() 1
```

If this is not defined, it is assumed that `MPID_Wtime` is not synchronized.

Packet **Packet**

Packet implementation - Describes the packets used the the Chameleon and Channel implementations of the ADI

Overview

The Chameleon and Channel implementations send message in *packets* containing either header information, data, or both.

The design here is fairly general, allowing both eager and reluctant (rendevous) protocols for each of the MPI modes. Some simplification can be achieved if, for example, the MPI Synchronous mode is only implemented with the reluctant protocol.

This is discussed in more detain in *Creating a new MPICH device using the Channel interface*, and is included here to give a quick summary of the approach.

Eager

In the eager protocol, the message data is sent without first asking the destination node (for example, to check on buffer space). In order to achieve better use of the underlying message-passing systems, there are two major categories: long and short. In a short message, the message data and message envelope (context, tag, lrank) are sent together. In a long message, the data is sent separately from the envelope (avoiding any data copies). If the MPI mode is Synchronous or Ready, then some additional information is included. In the synchronous case, it is necessary for the receiver to tell the sender when the receive has started (but see Rendevous below); this is done with the `MPID_PKT_SYNC_ACK`. In the ready case, if the matching receive is *not* available, then an error message can be sent back to the sender with `MPID_PKT_READY_ERROR`.

In the long message case, the *sender* may use nonblocking send operation. On some systems, it may be necessary to ask the sender to execute an MPI_Wait-like operation on the nonblocking send in order to get a matching receive to complete. On these systems, the macro `MPID_WSEND_BEFORE_WRECV` is defined, and the MPI_Wait on the send is requested with the `MPID_PKT_COMPLETE_SEND` packet. (The systems that require this tend to use rendevous or reluctant protocols for long messages). *THIS IS UNUSED.*

Rendevous

In the Rendevous protocol, a *long* send is not begun until the receiver oks it. The request is made with `MPID_PKT_REQUEST_SEND`, and is acknowledged with `MPID_PKT_OK_TO_SEND`. Note that a message-passing implementation may make use of a "ready-receiver" send once the `OK_TO_SEND` has been received. Also note that the rendevous protocol handles standard and synchronous sends identically; no separate `SYNC_ACK` packet is required.

Depending on the system, it may or may not be advantageous to compress the size of the data in the packet. If the macro `MPID_PKT_COMPRESSED` is defined, then the envelope data is encoded into 2 fewer ints.

Get

The *get* protocol is used on shared memory systems. The behavior of this protocol is very much like the *rendevous* protocol, except that the packets contain information needed to access the data directly from memory shared by the sender and the receiver (thus requiring a different packet type).

Threads **Threads**

Threads in the ADI - Support for multithreaded applications

MPI is thread-friendly in that the MPI specification does not specify anything that is not thread safe. However, the MPI specification does not mandate any particular thread support. It is our intent that the MPICH ADI provide a thread-safe version; this section details the current state of that interface. This is currently untested.

Communicator-based locks

Some operations on a communicator must be single threaded. These macros can be defined to provide the appropriate locks. The first parameter (`ctx`) is the ADI context (often ignored); the second is the communictor to lock.

MPID_THREAD_LOCK(ctx,comm)
: Lock a thread in a communicator

MPID_THREAD_UNLOCK(ctx,comm)
: Unlock a thread in a communicator

MPID_THREAD_LOCK_INIT(ctx,comm)
: Initialize a thread in a communicator

MPID_THREAD_LOCK_FINISH(ctx,comm)
: Free a thread in a communicator

Datastructure-based locks

These four are for locking individual data-structures. The data-structure should contain something like

```
typedef struct {
   MPID_THREAD_DS_LOCK_DECLARE
   other stuff
   } foo;
```

and then use

```
foo *p;
MPID_THREAD_DS_LOCK(p)
MPID_THREAD_DS_UNLOCK(p)
```

MPID_THREAD_DS_LOCK_DECLARE
: The declaration of the lock

MPID_THREAD_DS_LOCK_INIT(p)
: Initializes the lock for datastructure `p`

MPID_THREAD_DS_LOCK(p)
: Lock datastructure `p`

MPID_THREAD_DS_UNLOCK(p)
: Unlock datastructure `p`

Be very careful that thread locks are not nested or that nesting is handled carefully (for example, with a nest level).