

# 1 MPI Reference Card

<pre>int MPI_Abort( comm, errorcode ) MPI_Comm comm; int errorcode;</pre>	Terminates MPI execution environment
<pre>int MPI_Address( location, address) void *location; MPI_Aint *address;</pre>	Gets the address of a location in memory
<pre>int MPI_Allgatherv ( sendbuf, sendcount,                     sendtype, recvbuf, recvcnts,                     displs, recvtype, comm ) void *sendbuf; int sendcount; MPI_Datatype sendtype; void *recvbuf; int *recvcnts; int *displs; MPI_Datatype recvtype; MPI_Comm comm;</pre>	Gathers data from all tasks and deliver it to all
<pre>int MPI_Allgather ( sendbuf, sendcount,                    sendtype, recvbuf, recvcnt,                    recvtype, comm ) void *sendbuf; int sendcount; MPI_Datatype sendtype; void *recvbuf; int recvcnt; MPI_Datatype recvtype; MPI_Comm comm;</pre>	Gathers data from all tasks and distribute it to all
<pre>int MPI_Allreduce ( sendbuf, recvbuf,                    count, datatype, op, comm ) void *sendbuf; void *recvbuf; int count; MPI_Datatype datatype; MPI_Op op; MPI_Comm comm;</pre>	Combines values from all processes and distribute the result back to all processes
<pre>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;</pre>	Sends data from all to all processes, with a displacement

<pre>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;</pre>	Sends data from all to all processes
<pre>int MPI_Attr_delete ( comm, keyval ) MPI_Comm comm; int keyval;</pre>	Deletes attribute value associated with a key
<pre>int MPI_Attr_get ( comm, keyval,                  attr_value, flag ) MPI_Comm comm; int keyval; void *attr_value; int *flag;</pre>	Retrieves attribute value by key
<pre>int MPI_Attr_put ( comm, keyval,                  attr_value ) MPI_Comm comm; int keyval; void *attr_value;</pre>	Stores attribute value associated with a key
<pre>int MPI_Barrier ( comm ) MPI_Comm comm;</pre>	Blocks until all process have reached this routine.
<pre>int MPI_Bcast ( buffer, count, datatype,               root, comm ) void *buffer; int count; MPI_Datatype datatype; int root; MPI_Comm comm;</pre>	Broadcasts a message from the process with rank "root" to all other processes of the group.
<pre>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;</pre>	Builds a handle for a buffered send
<pre>int MPI_Bsend( buf, count, datatype,               dest, tag, comm ) void *buf; int count, dest, tag; MPI_Datatype datatype; MPI_Comm comm;</pre>	Basic send with user-specified buffering
<pre>int MPI_Buffer_attach( buffer, size ) void *buffer; int size;</pre>	Attaches a user-defined buffer for sending

<pre>int MPI_Buffer_detach( bufferptr, size ) void *bufferptr; int *size;</pre>	Removes an existing buffer (for use in MPI_Bsend etc)
<pre>int MPI_Cancel( request ) MPI_Request *request;</pre>	Cancels a communication request
<pre>int MPI_Cart_coords ( comm, rank,                     maxdims, coords ) MPI_Comm comm; int rank; int maxdims; int *coords;</pre>	Determines process coords in cartesian topology given rank in group
<pre>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;</pre>	Makes a new communicator to which topology information has been attached
<pre>int MPI_Cart_get ( comm, maxdims, dims,                  periods, coords ) MPI_Comm comm; int maxdims; int *dims, *periods, *coords;</pre>	Retrieves Cartesian topology information associated with a communicator
<pre>int MPI_Cart_map ( comm_old, ndims, dims,                  periods, newrank ) MPI_Comm comm_old; int ndims; int *dims; int *periods; int *newrank;</pre>	Maps process to Cartesian topology information
<pre>int MPI_Cart_rank ( comm, coords, rank ) MPI_Comm comm; int *coords; int *rank;</pre>	Determines process rank in communicator given Cartesian location
<pre>int MPI_Cart_shift ( comm, direction,                    displ, source, dest ) MPI_Comm comm; int direction; int displ; int *source; int *dest;</pre>	Returns the shifted source and destination ranks, given a shift direction and amount
<pre>int MPI_Cart_sub ( comm, remain_dims,                  comm_new ) MPI_Comm comm; int *remain_dims; MPI_Comm *comm_new;</pre>	Partitions a communicator into subgroups which form lower-dimensional cartesian subgrids
<pre>int MPI_Cartdim_get ( comm, ndims ) MPI_Comm comm; int *ndims;</pre>	Retrieves Cartesian topology information associated with a communicator

<pre>int MPI_Comm_compare ( comm1, comm2,                       result ) MPI_Comm comm1; MPI_Comm comm2; int *result;</pre>	Compares two communicators
<pre>int MPI_Comm_create ( comm, group,                      comm_out ) MPI_Comm comm; MPI_Group group; MPI_Comm *comm_out;</pre>	Creates a new communicator
<pre>int MPI_Comm_dup ( comm, comm_out ) MPI_Comm comm, *comm_out;</pre>	Duplicates an existing communicator with all its cached information
<pre>int MPI_Comm_free ( comm ) MPI_Comm *comm;</pre>	Marks the communicator object for deallocation
<pre>int MPI_Comm_group ( comm, group ) MPI_Comm comm; MPI_Group *group;</pre>	Accesses the group associated with given communicator
<pre>int MPI_Comm_rank ( comm, rank ) MPI_Comm comm; int *rank;</pre>	Determines the rank of the calling process in the communicator
<pre>int MPI_Comm_remote_group ( comm, group ) MPI_Comm comm; MPI_Group *group;</pre>	Accesses the remote group associated with the given inter-communicator
<pre>int MPI_Comm_remote_size ( comm, size ) MPI_Comm comm; int *size;</pre>	Determines the size of the remote group associated with an inter-communicator
<pre>int MPI_Comm_size ( comm, size ) MPI_Comm comm; int *size;</pre>	Determines the size of the group associated with a communicator
<pre>int MPI_Comm_split ( comm, color, key,                    comm_out ) MPI_Comm comm; int color, key; MPI_Comm *comm_out;</pre>	Creates new communicators based on colors and keys
<pre>int MPI_Comm_test_inter ( comm, flag ) MPI_Comm comm; int *flag;</pre>	Tests to see if a comm is an inter-communicator
<pre>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;</pre>	A function to simple-mindedly copy attributes
<pre>int MPI_Dims_create(nnodes, ndims, dims) int nnodes; int ndims; int *dims;</pre>	Creates a division of processors in a cartesian grid

<pre>int MPI_Errhandler_create( function,     errhandler ) MPI_Handler_function *function; MPI_Errhandler *errhandler;</pre>	Creates an MPI-style errorhandler
<pre>int MPI_Errhandler_free( errhandler ) MPI_Errhandler *errhandler;</pre>	Frees an MPI-style errorhandler
<pre>int MPI_Errhandler_get( comm, errhandler     ) MPI_Comm comm; MPI_Errhandler *errhandler;</pre>	Gets the error handler for a communicator
<pre>int MPI_Errhandler_set( comm, errhandler     ) MPI_Comm comm; MPI_Errhandler errhandler;</pre>	Sets the error handler for a communicator
<pre>int MPI_Error_class( errorcode,     errorclass ) int errorcode, *errorclass;</pre>	Converts an error code into an error class
<pre>int MPI_Error_string( errorcode, string,     resultlen ) int errorcode, *resultlen; char *string;</pre>	Return a string for a given error code
<pre>int MPI_Finalize()</pre>	Terminates MPI execution environment
<pre>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;</pre>	Gathers into specified locations from all processes in a group
<pre>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;</pre>	Gathers together values from a group of processes
<pre>int MPI_Get_count( status, datatype,     count ) MPI_Status *status; MPI_Datatype datatype; int *count;</pre>	Gets the number of "top level" elements

<pre>int MPI_Get_elements ( status, datatype,                       elements ) MPI_Status *status; MPI_Datatype datatype; int *elements;</pre>	Returns the number of basic elements in a datatype
<pre>int MPI_Get_processor_name( name,                             resultlen ) char *name; int *resultlen;</pre>	Gets the name of the processor
<pre>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;</pre>	Makes a new communicator to which topology information has been attached
<pre>int MPI_Graph_get ( comm, maxindex,                    maxedges, index, edges ) MPI_Comm comm; int maxindex, maxedges; int *index, *edges;</pre>	Retrieves graph topology information associated with a communicator
<pre>int MPI_Graph_map ( comm_old, nnodes,                    index, edges, newrank ) MPI_Comm comm_old; int nnodes; int *index; int *edges; int *newrank;</pre>	Maps process to graph topology information
<pre>int MPI_Graph_neighbors_count ( comm,                                 rank, nneighbors ) MPI_Comm comm; int rank; int *nneighbors;</pre>	Returns the number of neighbors of a node associated with a graph topology
<pre>int MPI_Graph_neighbors ( comm, rank,                           maxneighbors, neighbors ) MPI_Comm comm; int rank; int maxneighbors; int *neighbors;</pre>	Returns the neighbors of a node associated with a graph topology
<pre>int MPI_Graphdims_get ( comm, nnodes,                         nedges ) MPI_Comm comm; int *nnodes; int *nedges;</pre>	Retrieves graph topology information associated with a communicator
<pre>int MPI_Group_compare ( group1, group2,                        result ) MPI_Group group1; MPI_Group group2; int *result;</pre>	Compares two groups

<pre>int MPI_Group_difference ( group1,                           group2, group_out ) MPI_Group group1, group2, *group_out;</pre>	Makes a group from the difference of two groups
<pre>int MPI_Group_excl ( group, n, ranks,                     newgroup ) MPI_Group group, *newgroup; int n, *ranks;</pre>	Produces a group by reordering an existing group and taking only unlisted members
<pre>int MPI_Group_free ( group ) MPI_Group *group;</pre>	Frees a group
<pre>int MPI_Group_incl ( group, n, ranks,                     group_out ) MPI_Group group, *group_out; int n, *ranks;</pre>	Produces a group by reordering an existing group and taking only listed members
<pre>int MPI_Group_intersection ( group1,                              group2, group_out ) MPI_Group group1, group2, *group_out;</pre>	Produces a group as the intersection of two existing groups
<pre>int MPI_Group_range_excl ( group, n,                           ranges, newgroup ) MPI_Group group, *newgroup; int n, ranges[][3];</pre>	Produces a group by excluding ranges of processes from an existing group
<pre>int MPI_Group_range_incl ( group, n,                           ranges, newgroup ) MPI_Group group, *newgroup; int n, ranges[][3];</pre>	Creates a new group from ranges of ranks in an existing group
<pre>int MPI_Group_rank ( group, rank ) MPI_Group group; int *rank;</pre>	Returns the rank of this process in the given group
<pre>int MPI_Group_size ( group, size ) MPI_Group group; int *size;</pre>	Returns the size of a group
<pre>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;</pre>	Translates the ranks of processes in one group to those in another group
<pre>int MPI_Group_union ( group1, group2,                      group_out ) MPI_Group group1, group2, *group_out;</pre>	Produces a group by combining two groups
<pre>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;</pre>	Starts a nonblocking buffered send
<pre>int MPI_Initialized( flag ) int *flag;</pre>	Indicates whether 'MPI_Init' has been called.

<pre>int MPI_Init(argc,argv) int *argc; char ***argv;</pre>	Initialize the MPI execution environment
<pre>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;</pre>	Creates an intercommunicator from two intracommunicators
<pre>int MPI_Intercomm_merge ( comm, high,     comm_out ) MPI_Comm comm; int high; MPI_Comm *comm_out;</pre>	Creates an intracommunicator from an intercommunicator
<pre>int MPI_Iprobe( source, tag, comm, flag,     status ) int source; int tag; int *flag; MPI_Comm comm; MPI_Status *status;</pre>	Nonblocking test for a message
<pre>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;</pre>	Begins a nonblocking receive
<pre>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;</pre>	Starts a nonblocking ready send
<pre>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;</pre>	Begins a nonblocking send

<pre>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;</pre>	Starts a nonblocking synchronous send
<pre>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;</pre>	Generates a new attribute key
<pre>int MPI_Keyval_free ( keyval ) int *keyval;</pre>	Frees attribute key for communicator cache attribute
<pre>int MPI_Op_create( function, commute, op                  ) MPI_User_function *function; int commute; MPI_Op *op;</pre>	Creates a user-defined combination function handle
<pre>int MPI_Op_free( op ) MPI_Op *op;</pre>	Frees a user-defined combination function handle
<pre>int MPI_Pack_size ( incount, datatype,                   comm, size ) int incount; MPI_Datatype datatype; MPI_Comm comm; int *size;</pre>	Returns the upper bound on the amount of space needed to pack a message
<pre>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;</pre>	Packs a datatype into contiguous memory
<pre>int MPI_Pcontrol( level ) int level;</pre>	Controls profiling
<pre>int MPI_Probe( source, tag, comm, status              ) int source; int tag; MPI_Comm comm; MPI_Status *status;</pre>	Blocking test for a message

<pre>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;</pre>	Builds a handle for a receive
<pre>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;</pre>	Basic receive
<pre>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;</pre>	Combines values and scatters the results
<pre>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;</pre>	Reduces values on all processes to a single value
<pre>int MPI_Request_free( request ) MPI_Request *request;</pre>	Frees a communication request object
<pre>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;</pre>	Builds a handle for a ready send
<pre>int MPI_Rsend( buf, count, datatype,                dest, tag, comm ) void *buf; int count, dest, tag; MPI_Datatype datatype; MPI_Comm comm;</pre>	Basic ready send

<pre>int MPI_Scan ( sendbuf, recvbuf, count,               datatype, op, comm ) void *sendbuf; void *recvbuf; int count; MPI_Datatype datatype; MPI_Op op; MPI_Comm comm;</pre>	<p>Computes the scan (partial reductions) of data on a collection of processes</p>
<pre>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;</pre>	<p>Scatters a buffer in parts to all tasks in a group</p>
<pre>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;</pre>	<p>Sends data from one task to all other tasks in a group</p>
<pre>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;</pre>	<p>Builds a handle for a standard send</p>
<pre>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;</pre>	<p>Sends and receives using a single buffer</p>

<pre> 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; </pre>	Sends and receives a message
<pre> int MPI_Send( buf, count, datatype, dest,               tag, comm ) void *buf; int count, dest, tag; MPI_Datatype datatype; MPI_Comm comm; </pre>	Performs a basic send
<pre> 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; </pre>	Builds a handle for a synchronous send
<pre> int MPI_Ssend( buf, count, datatype,               dest, tag, comm ) void *buf; int count, dest, tag; MPI_Datatype datatype; MPI_Comm comm; </pre>	Basic synchronous send
<pre> int MPI_Startall( count,                  array_of_requests ) int count; MPI_Request array_of_requests[]; </pre>	Starts a collection of requests
<pre> int MPI_Start( request ) MPI_Request *request; </pre>	Initiates a communication with a persistent request handle
<pre> int MPI_Test_cancelled( status, flag ) MPI_Status *status; int *flag; </pre>	Tests to see if a request was canceled
<pre> 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; </pre>	Tests for the completion of all previously initiated communications

<pre>int MPI_Testany( count,                 array_of_requests, index, flag,                 status ) int count; MPI_Request array_of_requests[]; int *index, *flag; MPI_Status *status;</pre>	Tests for completion of any previously initiated communication
<pre>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[];</pre>	Tests for some given communications to complete
<pre>int MPI_Test ( request, flag, status ) MPI_Request *request; int *flag; MPI_Status *status;</pre>	Tests for the completion of a send or receive
<pre>int MPI_Topo_test ( comm, top_type ) MPI_Comm comm; int *top_type;</pre>	Determines the type of topology (if any) associated with a communicator
<pre>int MPI_Type_commit ( datatype ) MPI_Datatype *datatype;</pre>	Commits the datatype
<pre>int MPI_Type_contiguous( count, old_type,                         newtype ) int count; MPI_Datatype old_type; MPI_Datatype *newtype;</pre>	Creates a contiguous datatype
<pre>int MPI_Type_extent( datatype, extent ) MPI_Datatype datatype; MPI_Aint *extent;</pre>	Returns the extent of a datatype
<pre>int MPI_Type_free ( datatype ) MPI_Datatype *datatype;</pre>	Frees the datatype
<pre>int MPI_Type_hindexed( count, blocklens,                       indices, old_type, newtype ) int count; int blocklens[]; MPI_Aint indices[]; MPI_Datatype old_type; MPI_Datatype *newtype;</pre>	Creates an indexed datatype with offsets in bytes
<pre>int MPI_Type_hvector( count, blocklen,                     stride, old_type, newtype ) int count; int blocklen; MPI_Aint stride; MPI_Datatype old_type; MPI_Datatype *newtype;</pre>	Creates a vector (strided) datatype with offset in bytes

<pre>int MPI_Type_indexed( count, blocklens,                     indices, old_type, newtype ) int count; int blocklens[]; int indices[]; MPI_Datatype old_type; MPI_Datatype *newtype;</pre>	Creates an indexed datatype
<pre>int MPI_Type_lb ( datatype, displacement                 ) MPI_Datatype datatype; MPI_Aint *displacement;</pre>	Returns the lower-bound of a datatype
<pre>int MPI_Type_size ( datatype, size ) MPI_Datatype datatype; int *size;</pre>	Return the number of bytes occupied by entries in the datatype
<pre>int MPI_Type_struct( count, blocklens,                     indices, old_types, newtype ) int count; int blocklens[]; MPI_Aint indices[]; MPI_Datatype old_types[]; MPI_Datatype *newtype;</pre>	Creates a struct datatype
<pre>int MPI_Type_ub ( datatype, displacement                 ) MPI_Datatype datatype; MPI_Aint *displacement;</pre>	Returns the upper bound of a datatype
<pre>int MPI_Type_vector( count, blocklen,                     stride, old_type, newtype ) int count; int blocklen; int stride; MPI_Datatype old_type; MPI_Datatype *newtype;</pre>	Creates a vector (strided) datatype
<pre>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;</pre>	Unpack a datatype into contiguous memory
<pre>int MPI_Waitall(count, array_of_requests,                 array_of_statuses ) int count; MPI_Request array_of_requests[]; MPI_Status array_of_statuses[];</pre>	Waits for all given communications to complete
<pre>int MPI_Waitany(count, array_of_requests,                 index, status ) int count; MPI_Request array_of_requests[]; int *index; MPI_Status *status;</pre>	Waits for any specified send or receive to complete

<pre>int MPI_Wait( 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[];</pre>	<p>Waits for some given communications to complete</p>
<pre>int MPI_Wait ( request, status ) MPI_Request *request; MPI_Status *status;</pre>	<p>Waits for an MPI send or receive to complete</p>
<pre>double MPI_Wtick()</pre>	<p>Returns the resolution of MPI_Wtime</p>
<pre>double MPI_Wtime()</pre>	<p>Returns an elapsed time on the calling processor</p>