```
10,11c10,11
< The basic point-to-point communication operations are {\bf send} and
{\bf
< receive}.   Their use is illustrated in the example below.
---
> The basic point-to-point communication operations are \mpiterm{send}
and
> \mpiterm{receive}.   Their use is illustrated in the example below.
41c41
< In this example, process zero ({\sf myrank = 0}) sends a message to
process one
---
> In this example, process zero (\code{myrank = 0}) sends a message to
process one
43,44c43,44
< {\bf send} operation \mpifunc{MPI\_SEND}. The
< operation specifies a {\bf send buffer} in the sender memory from
which the
---
> \mpiterm{send} operation \mpifunc{MPI\_SEND}. The
> operation specifies a \mpiterm{send buffer} in the sender memory
from which the
46c46
< storage containing the variable {\sf message} in the memory of
process zero.
---
> storage containing the variable \mpiterm{message} in the memory of
process zero.
50c50
< In addition, the send operation associates an {\bf envelope} with
the
---
> In addition, the send operation associates an \mpiterm{envelope}
with the
52c52
< distinguishing information that can be used by the {\bf receive}
operation to
---
> distinguishing information that can be used by the \mpiterm{receive}
operation to
57,58c57,58
< Process one ({\sf myrank = 1}) receives this message with the
< {\bf receive} operation \mpifunc{MPI\_RECV}.
---
> Process one (\code{myrank = 1}) receives this message with the
> \mpiterm{receive} operation \mpifunc{MPI\_RECV}.
60c60
< envelope, and the message data is stored into the {\bf receive
---
> envelope, and the message data is stored into the \mpiterm{receive
```

```
62c62
< containing the string {\sf message} in the memory of process one.
---
> containing the string \code{message} in the memory of process one.
97c97
< \mpifnewbind{MPI\_Send(buf, count, datatype, dest, tag, comm,
ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: buf \\
INTEGER, INTENT(IN) :: count, dest, tag \\ TYPE(MPI\_Datatype),
INTENT(IN) :: datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Send(buf, count, datatype, dest, tag, comm,
ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: buf \\ INTEGER,
INTENT(IN) :: count, dest, tag \\ TYPE(MPI\_Datatype), INTENT(IN) ::
datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
108c108
< The send buffer specified by the \func{MPI\_SEND} operation consists
of
---
> The send buffer specified by the \mpifunc{MPI\_SEND} operation
consists of
144c144
< \caption{Predefined MPI datatypes corresponding to Fortran
datatypes}
---
> \caption{Predefined \MPI/ datatypes corresponding to Fortran
datatypes}
188c188
<                        & (defined in {\tt <stddef.h>}) \\
---
>                        & (defined in \code{<stddef.h>}) \\
209c209
< \caption{Predefined MPI datatypes corresponding to C datatypes}
---
> \caption{Predefined \MPI/ datatypes corresponding to C datatypes}
225c225
< \caption{Predefined MPI datatypes corresponding to both C and
Fortran datatypes}
---
> \caption{Predefined \MPI/ datatypes corresponding to both C and
Fortran datatypes}
303c303
< \caption{Predefined MPI datatypes corresponding to C++ datatypes}
---
> \caption{Predefined \MPI/ datatypes corresponding to C++ datatypes}
314c314
< the {\bf message envelope}.   These fields are
---
> the \mpiterm{message envelope}.   These fields are
```

```
332c332,333
< The range of valid tag values is {\sf 0,...,UB}, where the value of
{\sf UB} is
---
> The range of valid tag values is $0,\ldots,\mpicode{UB}$, where the
value of
> \mpicode{UB} is
335,336c336,337
< described in Chapter~\ref{chap:environment}.  \MPI/ requires that
{\sf UB} be no
< less than 32767.
---
> described in Chapter~\ref{chap:environment}.  \MPI/ requires that
> \mpicode{UB} be no less than 32767.
338c339
< The \mpiarg{comm} argument specifies the {\bf communicator} that is
used for
---
> The \mpiarg{comm} argument specifies the \mpiterm{communicator} that
is used for
350c351
< communication context.   This {\bf process group}
---
> communication context.   This \mpiterm{process group}
353c354
< {\sf 0, ..., n-1}$\cup$\{\const{MPI\_PROC\_NULL}\}, where {\sf n} is
the number of
---
> $0, \ldots, n-1 \cup \{\const{MPI\_PROC\_NULL}\}$, where $n$ is the
number of
408c409
< \mpifnewbind{MPI\_Recv(buf, count, datatype, source, tag, comm,
status, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..) :: buf \\
INTEGER, INTENT(IN) :: count, source, tag \\ TYPE(MPI\_Datatype),
INTENT(IN) :: datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\
TYPE(MPI\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) ::
ierror}
---
> \mpifnewbind{MPI\_Recv(buf, count, datatype, source, tag, comm,
status, ierror) \fargs TYPE(*), DIMENSION(..) :: buf \\ INTEGER,
INTENT(IN) :: count, source, tag \\ TYPE(MPI\_Datatype), INTENT(IN) ::
datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\ TYPE(MPI
\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
476,477c477,478
< \{{\sf 0,...,n-1}\}$\cup$\{\const{MPI\_ANY\_SOURCE}\},$\cup$\
{\const{MPI\_PROC\_NULL}\}, where
< {\sf n} is the number of processes in this group.
---
> \{$0,\ldots,n-1\}\cup\{\const{MPI\_ANY\_SOURCE}\},\cup\{\const{MPI
\_PROC\_NULL}\}$, where
```

```
> $n$ is the number of processes in this group.
502c503
< Section~\ref{sec:pt2pt-nullproc} on page~\pageref{sec:pt2pt-
nullproc}.
---
> \sectionref{sec:pt2pt-nullproc}.
527c528
< In Fortran with {\tt USE} {\tt mpi} or {\tt INCLUDE} {\tt 'mpif.h'},
---
> In Fortran with \code{USE} \code{mpi} or \code{INCLUDE}
\code{'mpif.h'},
537c538
< With Fortran {\tt USE} {\tt mpi\_f08}, status is defined as the
---
> With Fortran \code{USE} \code{mpi\_f08}, status is defined as the
539c540
< containing three public fields named \const{MPI\_SOURCE},
---
> containing three public \ftype{INTEGER} fields named \const{MPI
\_SOURCE},
542,543c543,544
< Thus, {\tt status\%MPI\_SOURCE}, {\tt status\%MPI\_TAG}
< and {\tt status\%MPI\_ERROR} contain the source,
---
> Thus, \code{status\%MPI\_SOURCE}, \code{status\%MPI\_TAG}
> and \code{status\%MPI\_ERROR} contain the source,
545c546
< Additionally, within both the {\tt mpi} and the {\tt mpi\_f08}
modules,
---
> Additionally, within both the \code{mpi} and the \code{mpi\_f08}
modules,
551c552
< Section~\ref{sec:conversion:status} on page~
\pageref{sec:conversion:status}.
---
> \sectionref{sec:conversion:status}.
582c583
< status, such as \func{MPI\_WAIT}, since that would only duplicate
the
---
> status, such as \mpifunc{MPI\_WAIT}, since that would only duplicate
the
604c605
< \mpifnewbind{MPI\_Get\_count(status, datatype, count, ierror)
BIND(C) \fargs TYPE(MPI\_Status), INTENT(IN) :: status \\ TYPE(MPI
\_Datatype), INTENT(IN) :: datatype \\ INTEGER, INTENT(OUT) :: count \
\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Get\_count(status, datatype, count, ierror) \fargs
TYPE(MPI\_Status), INTENT(IN) :: status \\ TYPE(MPI\_Datatype),
INTENT(IN) :: datatype \\ INTEGER, INTENT(OUT) :: count \\ INTEGER,
OPTIONAL, INTENT(OUT) :: ierror}
613c614
< parameter, then \func{MPI\_GET\_COUNT} sets the value of
\mpiarg{count} to
---
> parameter, then \mpifunc{MPI\_GET\_COUNT} sets the value of
\mpiarg{count} to
630c631
< The \mpiarg{datatype} argument is passed to \func{MPI\_GET\_COUNT}
so as to
---
> The \mpiarg{datatype} argument is passed to \mpifunc{MPI\_GET
\_COUNT} so as to
634,635c635,636
< \func{MPI\_PROBE} or \func{MPI\_IPROBE}. With a status from
\func{MPI\_PROBE} or \func{MPI\_IPROBE},
< the same datatypes are allowed as in a call to \func{MPI\_RECV} to
receive this message.
---
> \mpifunc{MPI\_PROBE} or \mpifunc{MPI\_IPROBE}. With a status from
\mpifunc{MPI\_PROBE} or \mpifunc{MPI\_IPROBE},
> the same datatypes are allowed as in a call to \mpifunc{MPI\_RECV}
to receive this message.
659c660
< In most cases, the safest approach is to use the same datatype with
\func{MPI\_GET\_COUNT} and the receive.
---
> In most cases, the safest approach is to use the same datatype with
\mpifunc{MPI\_GET\_COUNT} and the receive.
665,666c666,667
< \func{MPI\_SEND} and
< \func{MPI\_RECV} operations described in this section.
---
> \mpifunc{MPI\_SEND} and
> \mpifunc{MPI\_RECV} operations described in this section.
671,672d671
< \status{Passed twice.}
<
679c678
< in {\tt mpi.h} and {\tt mpif.h}, and it exists in the user's
program.  In many
---
> in \code{mpi.h} and \code{mpif.h}, and it exists in the user's
program.  In many
681c680
< to examine the {\tt status} fields.  In these cases, it is a waste
for the user
---
```

```
> to examine the \texttt{status} fields.  In these cases, it is a
waste for the user
718c717
< \mpiskipfunc{{\sf MPI\_}\textrm{\{}{\sf TEST$|$WAIT}\textrm{\}\{}
{\sf ALL$|$SOME}\textrm{\}}}
---
> \mpiskipfunc{\mpicode{MPI\_}\textrm{\{}\mpicode{TEST$|$WAIT}
\textrm{\}\{}\mpicode{ALL$|$SOME}\textrm{\}}}
731c730
< \mpiskipfunc{{\sf MPI\_}\textrm{\{}{\sf TEST$|$WAIT}\textrm{\}\{}
{\sf ALL$|$SOME}\textrm{\}}}
---
> \mpiskipfunc{\mpicode{MPI\_}\textrm{\{}\mpicode{TEST$|$WAIT}
\textrm{\}\{}\mpicode{ALL$|$SOME}\textrm{\}}}
819c818
< \begin{example} {\rm
---
> \begin{example}
840c839
< This code is correct if both {\sf a} and {\sf b} are real arrays of
---
> This code is correct if both \code{a} and \code{b} are real arrays
of
842c841
< even if {\sf a} or {\sf b} have size $< 10$: e.g., when {\sf a(1)}
can
---
> even if \code{a} or \code{b} have size $< 10$: e.g., when
\code{a(1)} can
844c843
< } \end{example}
---
> \end{example}
846c845
< \begin{example} {\rm
---
> \begin{example}
870c869
< } \end{example}
---
> \end{example}
872c871
< \begin{example} {\rm
---
> \begin{example}
894,896c893,895
< This code is correct, irrespective of the type and size of {\sf a}
and
< {\sf b} (unless this results in an out of bounds memory access).
< } \end{example}
```

```
---
> This code is correct, irrespective of the type and size of \code{a}
and
> \code{b} (unless this results in an out of bounds memory access).
> \end{example}
923c922
< \begin{example} {\rm
---
> \begin{example}
951c950
< } \end{example}
---
> \end{example}
1028c1027
< The first program is correct, assuming that {\sf a} and {\sf b} are
---
> The first program is correct, assuming that \code{a} and \code{b}
are
1047c1046
< representation as the message received.  If {\sf a} and {\sf b} are
---
> representation as the message received.  If \code{a} and \code{b}
are
1076c1075
< Section~\ref{sec:misc-lang-interop} on page~\pageref{sec:misc-lang-
interop}.
---
> \sectionref{sec:misc-lang-interop}.
1082c1081
< is {\bf blocking}:
---
> is \mpiterm{blocking}:
1102c1101
< the {\bf standard} communication mode.  In this mode,
---
> the \mpiterm{standard} communication mode.  In this mode,
1115c1114
< standard mode send is {\bf non-local}:  successful completion of the
send
---
> standard mode send is \mpiterm{non-local}:  successful completion of
the send
1134c1133
< A {\bf buffered} mode send operation can be started whether or not a
---
> A \mpiterm{buffered} mode send operation can be started whether or
not a
1137c1136
< the standard send, this operation is {\bf local}, and its
---
```

```
> the standard send, this operation is \mpiterm{local}, and its
1146c1145
< A send that uses the {\bf synchronous} mode can be started whether
or
---
> A send that uses the \mpiterm{synchronous} mode can be started
whether or
1160c1159
< communication.  A send executed in this mode is {\bf non-local}.
---
> communication.  A send executed in this mode is \mpiterm{non-local}.
1162c1161
< A send that uses the {\bf ready} communication mode
---
> A send that uses the \mpiterm{ready} communication mode
1181,1183c1180,1182
< {\sf B} for buffered,
< {\sf S} for synchronous, and
< {\sf R} for ready.
---
> \mpicode{B} for buffered,
> \mpicode{S} for synchronous, and
> \mpicode{R} for ready.
1200c1199
< \mpifnewbind{MPI\_Bsend(buf, count, datatype, dest, tag, comm,
ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: buf \\
INTEGER, INTENT(IN) :: count, dest, tag \\ TYPE(MPI\_Datatype),
INTENT(IN) :: datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Bsend(buf, count, datatype, dest, tag, comm,
ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: buf \\ INTEGER,
INTENT(IN) :: count, dest, tag \\ TYPE(MPI\_Datatype), INTENT(IN) ::
datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
1219c1218
< \mpifnewbind{MPI\_Ssend(buf, count, datatype, dest, tag, comm,
ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: buf \\
INTEGER, INTENT(IN) :: count, dest, tag \\ TYPE(MPI\_Datatype),
INTENT(IN) :: datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Ssend(buf, count, datatype, dest, tag, comm,
ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: buf \\ INTEGER,
INTENT(IN) :: count, dest, tag \\ TYPE(MPI\_Datatype), INTENT(IN) ::
datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
1238c1237
< \mpifnewbind{MPI\_Rsend(buf, count, datatype, dest, tag, comm,
ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: buf \\
INTEGER, INTENT(IN) :: count, dest, tag \\ TYPE(MPI\_Datatype),
INTENT(IN) :: datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Rsend(buf, count, datatype, dest, tag, comm,
ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: buf \\ INTEGER,
INTENT(IN) :: count, dest, tag \\ TYPE(MPI\_Datatype), INTENT(IN) ::
datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
1247c1246
< The receive operation described in the last section is {\bf
blocking}:
---
> The receive operation described in the last section is
\mpiterm{blocking}:
1276c1275
< {\sf ready send}:   The message is sent as soon as possible.
---
> \mpiterm{ready send}:   The message is sent as soon as possible.
1278c1277
< {\sf synchronous send:}
---
> \mpiterm{synchronous send}:
1284c1283
< {\sf standard send:}
---
> \mpiterm{standard send}:
1288c1287
< {\sf buffered send:}
---
> \mpiterm{buffered send}:
1342c1341
< \begin{example} {\rm
---
> \begin{example}
1367c1366
< } \end{example}
---
> \end{example}
1381c1380
< \begin{example} {\rm
---
> \begin{example}
1415c1414
< } \end{example}
---
> \end{example}
1474c1473
< \begin{example} {\rm
---
```

```
> \begin{example}
1501c1500
< } \end{example}
---
> \end{example}
1503c1502
< \begin{example} {\rm
---
> \begin{example}
1508,1510c1507
< An
< errant
< attempt to exchange messages.
---
> An errant attempt to exchange messages.
1534c1531
< } \end{example}
---
> \end{example}
1536c1533
< \begin{example} {\rm
---
> \begin{example}
1565c1562
< } \end{example}
---
> \end{example}
1622c1619
< \mpifnewbind{MPI\_Buffer\_attach(buffer, size, ierror) BIND(C)
\fargs TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buffer \\ INTEGER,
INTENT(IN) :: size \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Buffer\_attach(buffer, size, ierror) \fargs
TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buffer \\ INTEGER,
INTENT(IN) :: size \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
1633c1630
< Section~\ref{sec:misc-sequence} on page~\pageref{sec:misc-
sequence}).
---
> \sectionref{sec:misc-sequence}.
1644c1641
< \mpifnewbind{MPI\_Buffer\_detach(buffer\_addr, size, ierror) BIND(C)
\fargs USE, INTRINSIC :: ISO\_C\_BINDING, ONLY : C\_PTR \\ TYPE(C
\_PTR), INTENT(OUT) :: buffer\_addr \\ INTEGER, INTENT(OUT) :: size \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Buffer\_detach(buffer\_addr, size, ierror) \fargs
USE, INTRINSIC :: ISO\_C\_BINDING, ONLY : C\_PTR \\ TYPE(C\_PTR),
INTENT(OUT) :: buffer\_addr \\ INTEGER, INTENT(OUT) :: size \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
```

```
1654c1651
< \begin{example}{\rm
---
> \begin{example}
1674c1671
< } \end{example}
---
> \end{example}
1683c1680
< In Fortran with the {\tt mpi} module or {\tt mpif.h}, the type of
the \mpiarg{buffer\_addr} argument is
---
> In Fortran with the \code{mpi} module or \code{mpif.h}, the type of
the \mpiarg{buffer\_addr} argument is
1685c1682
< In Fortran with the {\tt mpi\_f08} module, the address of the buffer
is returned
---
> In Fortran with the \code{mpi\_f08} module, the address of the
buffer is returned
1687c1684
< Example~\ref{ex:1side-fmalloc-ptr} on page~\pageref{ex:1side-
fmalloc-ptr}
---
> \namedref{Example}{ex:1side-fmalloc-ptr}
1759,1761c1756,1758
< Compute the number, {\sf n}, of bytes needed to store an entry for
the new message.
< An upper bound on {\sf n} can be computed
< as follows: A call to the function \linebreak
---
> Compute the number, $n$, of bytes needed to store an entry for the
new message.
> An upper bound on $n$ can be computed
> as follows: A call to the function\flushline
1763c1760
< comm, size)}, with the \texttt{count}, \texttt{datatype} and {\tt
comm} arguments
---
> comm, size)}, with the \mpicode{count}, \mpicode{datatype} and
\mpicode{comm} arguments
1772c1769
< Find the next contiguous empty space of {\sf n} bytes in
---
> Find the next contiguous empty space of $n$ bytes in
1796,1797c1793,1794
< better performance is to use {\bf nonblocking communication}.  A
< nonblocking {\bf send start} call initiates the send operation, but
does not
---
```

```
> better performance is to use \mpiterm{nonblocking communication}.  A
> nonblocking \mpiterm{send start} call initiates the send operation,
but does not
1801c1798
< A separate {\bf send complete}
---
> A separate \mpiterm{send complete}
1807c1804
< Similarly, a nonblocking {\bf receive start call} initiates the
receive
---
> Similarly, a nonblocking \mpiterm{receive start call} initiates the
receive
1811c1808
< a message is stored into the receive buffer.  A separate {\bf
receive
---
> a message is stored into the receive buffer.  A separate
\mpiterm{receive
1822c1819,1820
< {\sf standard}, {\sf buffered}, {\sf synchronous} and {\sf ready}.
These carry
---
> \mpiterm{standard}, \mpiterm{buffered}, \mpiterm{synchronous} and
> \mpiterm{ready}.  These carry
1824,1825c1822,1823
< Sends of all modes, {\sf ready} excepted, can be started whether a
matching
< receive has been posted or not; a nonblocking {\sf ready}
---
> Sends of all modes, \mpiterm{ready} excepted, can be started whether
a matching
> receive has been posted or not; a nonblocking \mpiterm{ready}
1841c1839
< If the send mode is {\sf synchronous}, then the
---
> If the send mode is \mpiterm{synchronous}, then the
1851c1849
< If the send mode is {\sf buffered} then the
---
> If the send mode is \mpiterm{buffered} then the
1857c1855
< If the send mode is {\sf standard} then the send-complete call may
---
> If the send mode is \mpiterm{standard} then the send-complete call
may
1899c1897
< Nonblocking communications use opaque {\sf request} objects to
---
> Nonblocking communications use opaque \mpiterm{request} objects to
```

```
1914,1916c1912,1915
< prefix of {\sf B}, {\sf S}, or {\sf R} is used for {\sf buffered},
{\sf
< synchronous} or {\sf ready} mode.  In addition a prefix of {\sf I}
(for {\sf
< immediate}) indicates that the call is nonblocking.
---
> prefix of \mpicode{B}, \mpicode{S}, or \mpicode{R} is used for
> \mpiterm{buffered}, \mpiterm{synchronous} or \mpiterm{ready} mode.
> In addition a prefix of \mpicode{I} (for \mpiterm{immediate})
indicates
> that the call is nonblocking.
1933c1932
< \mpifnewbind{MPI\_Isend(buf, count, datatype, dest, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Isend(buf, count, datatype, dest, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
1956c1955
< \mpifnewbind{MPI\_Ibsend(buf, count, datatype, dest, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Ibsend(buf, count, datatype, dest, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
1980c1979
< \mpifnewbind{MPI\_Issend(buf, count, datatype, dest, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Issend(buf, count, datatype, dest, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
2003c2002
< \mpifnewbind{MPI\_Irsend(buf, count, datatype, dest, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Irsend(buf, count, datatype, dest, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
2026c2025
< \mpifnewbind{MPI\_Irecv(buf, count, datatype, source, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, source, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Irecv(buf, count, datatype, source, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf \\
INTEGER, INTENT(IN) :: count, source, tag \\ TYPE(MPI\_Datatype),
INTENT(IN) :: datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\
TYPE(MPI\_Request), INTENT(OUT) :: request \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
2050a2050,2052
> %%
> %% See comments elsewhere in this file as to why the commented out
language
> %% is both incorrect and unnecessary.
2054,2063c2056,2065
< Sections~\ref{sec:misc-problems}-\ref{sec:f90-problems:comparison-
with-C},
< especially in
< Sections~\ref{sec:misc-sequence} and~\ref{sec:f90-problems:vector-
subscripts}
< on pages~\pageref{sec:misc-sequence}-\pageref{sec:f90-
problems:vector-subscripts}
< about ``{\sf Problems Due to Data Copying and Sequence Association
with Subscript Triplets}''
< and ``{\sf Vector Subscripts}'',
< and in Sections~\ref{sec:misc-register} to~\ref{sec:f90-
problems:perm-data-movements}
```

```
< on pages~\pageref{sec:misc-register} to~\pageref{sec:f90-
problems:perm-data-movements}
< about ``{\sf Optimization Problems}'', ``{\sf Code Movements and
Register Optimization}'',
< ``{\sf Temporary Data Movements}'' and ``{\sf Permanent Data
Movements}''.
---
> Sections~\ref{sec:misc-problems}--\ref{sec:f90-problems:comparison-
with-C}.
> %% especially in
> %% Sections~\ref{sec:misc-sequence} and~\ref{sec:f90-
problems:vector-subscripts}
> %% on pages~\pageref{sec:misc-sequence}-\pageref{sec:f90-
problems:vector-subscripts}
> %% about ``{\sf Problems Due to Data Copying and Sequence
Association with Subscript Triplets}''
> %% and ``{\sf Vector Subscripts}'',
> %% and in Sections~\ref{sec:misc-register} to~\ref{sec:f90-
problems:perm-data-movements}
> %% on pages~\pageref{sec:misc-register} to~\pageref{sec:f90-
problems:perm-data-movements}
> %% about ``{\sf Optimization Problems}'', ``{\sf Code Movements and
Register Optimization}'',
> %% ``{\sf Temporary Data Movements}'' and ``{\sf Permanent Data
Movements}''.
2070c2072
< The functions \func{MPI\_WAIT} and \func{MPI\_TEST} are used to
complete a
---
> The functions \mpifunc{MPI\_WAIT} and \mpifunc{MPI\_TEST} are used
to complete a
2078c2080
< subsystem.  However, if a {\sf synchronous}
---
> subsystem.  However, if a \mpiterm{synchronous}
2090,2091c2092,2094
< A {\bf null} handle is a handle with
< value \linebreak \const{MPI\_REQUEST\_NULL}.
---
> A \mpiterm{null} handle is a handle with
> value\flushline
> \const{MPI\_REQUEST\_NULL}.
2093c2096
< request and the handle to it are {\bf inactive}
---
> request and the handle to it are \mpiterm{inactive}
2095,2096c2098,2099
< communication (see Section~\ref{sec:pt2pt-persistent}).
< A handle is {\bf active} if it is neither null nor inactive.
---
```

```
> communication (see \sectionref{sec:pt2pt-persistent}).
> A handle is \mpiterm{active} if it is neither null nor inactive.
2098c2101
< {\bf empty} status is a status which is set to return \mpiarg{tag =}
---
> \mpiterm{empty} status is a status which is set to return
\mpiarg{tag =}
2101c2104
< \mpifunc{MPI\_GET\_COUNT}, \mpifunc{MPI\_GET\_ELEMENTS}, and
\func{MPI\_GET\_ELEMENTS\_X} return
---
> \mpifunc{MPI\_GET\_COUNT}, \mpifunc{MPI\_GET\_ELEMENTS}, and
\mpifunc{MPI\_GET\_ELEMENTS\_X} return
2110c2113
< ({\sf MPI\_}\{{\sf TEST$|$WAIT}\}\{{\sf ALL$|$SOME$|$ANY}\}),
---
> (\mpicode{MPI\_}\{\mpicode{TEST$|$WAIT}\}\{\mpicode{ALL$|$SOME$|
$ANY}\}),
2133c2136
< \mpifnewbind{MPI\_Wait(request, status, ierror) BIND(C) \fargs
TYPE(MPI\_Request), INTENT(INOUT) :: request \\ TYPE(MPI\_Status) ::
status \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Wait(request, status, ierror) \fargs TYPE(MPI
\_Request), INTENT(INOUT) :: request \\ TYPE(MPI\_Status) :: status \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
2138c2141
< A call to \func{MPI\_WAIT} returns when the operation
---
> A call to \mpifunc{MPI\_WAIT} returns when the operation
2158c2161
< Successful return of \func{MPI\_WAIT} after a \func{MPI\_IBSEND}
implies
---
> Successful return of \mpifunc{MPI\_WAIT} after a \mpifunc{MPI
\_IBSEND} implies
2161c2164
< buffer attached with \func{MPI\_BUFFER\_ATTACH}.   Note that, at
this point,
---
> buffer attached with \mpifunc{MPI\_BUFFER\_ATTACH}.   Note that, at
this point,
2165c2168
< counter to the stated goal of \func{MPI\_CANCEL} (always being able
to free
---
> counter to the stated goal of \mpifunc{MPI\_CANCEL} (always being
able to free
2171c2174
< \func{MPI\_WAIT} should block only the calling thread, allowing the
```

```
thread
---
> \mpifunc{MPI\_WAIT} should block only the calling thread, allowing
the thread
2177c2180
< \funcarg{\OUT}{flag}{{\sf true} if operation completed (logical)}
---
> \funcarg{\OUT}{flag}{\mpicode{true} if operation completed
(logical)}
2185c2188
< \mpifnewbind{MPI\_Test(request, flag, status, ierror) BIND(C) \fargs
TYPE(MPI\_Request), INTENT(INOUT) :: request \\ LOGICAL,
INTENT(OUT) :: flag \\ TYPE(MPI\_Status) :: status \\ INTEGER,
OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Test(request, flag, status, ierror) \fargs
TYPE(MPI\_Request), INTENT(INOUT) :: request \\ LOGICAL,
INTENT(OUT) :: flag \\ TYPE(MPI\_Status) :: status \\ INTEGER,
OPTIONAL, INTENT(OUT) :: ierror}
2192c2195
< A call to \func{MPI\_TEST} returns \mpiarg{flag = true} if the
---
> A call to \mpifunc{MPI\_TEST} returns \mpiarg{flag = true} if the
2202c2205
< \func{MPI\_TEST} is a local operation.
---
> \mpifunc{MPI\_TEST} is a local operation.
2217c2220
< The functions \func{MPI\_WAIT} and \func{MPI\_TEST} can be used to
---
> The functions \mpifunc{MPI\_WAIT} and \mpifunc{MPI\_TEST} can be
used to
2222c2225
< the nonblocking \func{MPI\_TEST} call allows the user to
---
> the nonblocking \mpifunc{MPI\_TEST} call allows the user to
2229c2232
< \begin{example} {\rm
---
> \begin{example}
2235c2238
< Simple usage of nonblocking operations and \func{MPI\_WAIT}.
---
> Simple usage of nonblocking operations and \mpifunc{MPI\_WAIT}.
2256c2259
< } \end{example}
---
> \end{example}
2268c2271
< \mpifnewbind{MPI\_Request\_free(request, ierror) BIND(C) \fargs
TYPE(MPI\_Request), INTENT(INOUT) :: request \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Request\_free(request, ierror) \fargs TYPE(MPI
\_Request), INTENT(INOUT) :: request \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
2293c2296
< \begin{example} {\rm
---
> \begin{example}
2302d2304
<
2331c2333
< } \end{example}
---
> \end{example}
2347c2349
< \begin{example} {\rm
---
> \begin{example}
2375c2377
< } \end{example}
---
> \end{example}
2378c2380
< A call to \func{MPI\_WAIT} that completes a receive will eventually
---
> A call to \mpifunc{MPI\_WAIT} that completes a receive will
eventually
2384c2386
< sender to complete the send.   Similarly, a call to \func{MPI\_WAIT}
that
---
> sender to complete the send.   Similarly, a call to \mpifunc{MPI
\_WAIT} that
2391c2393
< \begin{example} {\rm
---
> \begin{example}
2423c2425
< } \end{example}
---
> \end{example}
2425c2427
< If an \func{MPI\_TEST} that completes a receive is repeatedly called
---
> If an \mpifunc{MPI\_TEST} that completes a receive is repeatedly
called
2431c2433
< If an \func{MPI\_TEST} that completes a send is repeatedly called
```

```
with the
---
> If an \mpifunc{MPI\_TEST} that completes a send is repeatedly called
with the
2442c2444
< A call to \func{MPI\_WAITANY} or \func{MPI\_TESTANY} can be used to
---
> A call to \mpifunc{MPI\_WAITANY} or \mpifunc{MPI\_TESTANY} can be
used to
2444,2445c2446,2447
< completion of one out of several operations. A call to \func{MPI
\_WAITALL}
< or \func{MPI\_TESTALL} can be
---
> completion of one out of several operations. A call to \mpifunc{MPI
\_WAITALL}
> or \mpifunc{MPI\_TESTALL} can be
2463c2465
< \mpifnewbind{MPI\_Waitany(count, array\_of\_requests, index, status,
ierror) BIND(C) \fargs INTEGER, INTENT(IN) :: count \\ TYPE(MPI
\_Request), INTENT(INOUT) :: array\_of\_requests(count) \\ INTEGER,
INTENT(OUT) :: index \\ TYPE(MPI\_Status) :: status \\ INTEGER,
OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Waitany(count, array\_of\_requests, index, status,
ierror) \fargs INTEGER, INTENT(IN) :: count \\ TYPE(MPI\_Request),
INTENT(INOUT) :: array\_of\_requests(count) \\ INTEGER, INTENT(OUT) ::
index \\ TYPE(MPI\_Status) :: status \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
2489c2491
< The execution of \func{MPI\_WAITANY(count, array\_of\_requests,
index,
---
> The execution of \mpifunc{MPI\_WAITANY(count, array\_of\_requests,
index,
2491c2493
< \func{MPI\_WAIT(\&array\_of\_requests[i], status)},
---
> \mpifunc{MPI\_WAIT(\&array\_of\_requests[i], status)},
2515c2517
< \mpifnewbind{MPI\_Testany(count, array\_of\_requests, index, flag,
status, ierror) BIND(C) \fargs INTEGER, INTENT(IN) :: count \\
TYPE(MPI\_Request), INTENT(INOUT) :: array\_of\_requests(count) \\
INTEGER, INTENT(OUT) :: index \\ LOGICAL, INTENT(OUT) :: flag \\
TYPE(MPI\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) ::
ierror}
---
> \mpifnewbind{MPI\_Testany(count, array\_of\_requests, index, flag,
status, ierror) \fargs INTEGER, INTENT(IN) :: count \\ TYPE(MPI
\_Request), INTENT(INOUT) :: array\_of\_requests(count) \\ INTEGER,
INTENT(OUT) :: index \\ LOGICAL, INTENT(OUT) :: flag \\ TYPE(MPI
\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
2523,2525c2525,2527
< In the former case, it returns {\sf flag = true},
< returns in {\sf index} the index of this request in the array,
< and returns in {\sf status} the status of that operation.  If
---
> In the former case, it returns \mpicode{flag = true},
> returns in \mpicode{index} the index of this request in the array,
> and returns in \mpicode{status} the status of that operation.  If
2531c2533
< In the latter case (no operation completed), it returns {\sf flag =
---
> In the latter case (no operation completed), it returns
\mpicode{flag =
2539c2541
< immediately with {\sf flag = true},
---
> immediately with \mpicode{flag = true},
2543c2545
< the execution of \func{MPI\_TESTANY(count, array\_of\_requests,
---
> the execution of \mpifunc{MPI\_TESTANY(count, array\_of\_requests,
2545,2548c2547,2551
< \func{MPI\_TEST( \&array\_of\_requests[i], flag, status)},
< for {\sf i=0, 1 ,..., count-1},
< in some arbitrary order, until one call returns {\sf flag = true},
or
< all fail.  In the former case, {\sf index} is set to the last value
of {\sf i},
---
> \mpifunc{MPI\_TEST( \&array\_of\_requests[i], flag, status)},
> for \mpicode{i=0, 1 ,$\ldots$, count-1},
> in some arbitrary order, until one call returns \mpicode{flag =
true}, or
> all fail.  In the former case, \mpicode{index} is set to the last
value of
> \mpicode{i},
2567c2570
< \mpifnewbind{MPI\_Waitall(count, array\_of\_requests, array\_of
\_statuses, ierror) BIND(C) \fargs INTEGER, INTENT(IN) :: count \\
TYPE(MPI\_Request), INTENT(INOUT) :: array\_of\_requests(count) \\
TYPE(MPI\_Status) :: array\_of\_statuses(*) \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Waitall(count, array\_of\_requests, array\_of
\_statuses, ierror) \fargs INTEGER, INTENT(IN) :: count \\ TYPE(MPI
\_Request), INTENT(INOUT) :: array\_of\_requests(count) \\ TYPE(MPI
\_Status) :: array\_of\_statuses(*) \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
```

```
2586c2589
< The error-free execution of \func{MPI\_WAITALL(count, array\_of
\_requests,
---
> The error-free execution of \mpifunc{MPI\_WAITALL(count, array\_of
\_requests,
2589,2590c2592,2593
< \func{MPI\_WAIT(\&array\_of\_request[i], \&array\_of\_statuses[i])},
< for {\sf i=0 ,..., count-1}, in some arbitrary order.
---
> \mpifunc{MPI\_WAIT(\&array\_of\_request[i], \&array\_of
\_statuses[i])},
> for \mpicode{i=0 ,$\ldots$, count-1}, in some arbitrary order.
2595c2598
< call to \func{MPI\_WAITALL} fail, it is
---
> call to \mpifunc{MPI\_WAITALL} fail, it is
2597c2600
< communication.  The function \func{MPI\_WAITALL} will return in such
---
> communication.  The function \mpifunc{MPI\_WAITALL} will return in
such
2603c2606
< The function \func{MPI\_WAITALL} will return \const{MPI\_SUCCESS} if
no request
---
> The function \mpifunc{MPI\_WAITALL} will return \const{MPI\_SUCCESS}
if no request
2631c2634
< \mpifnewbind{MPI\_Testall(count, array\_of\_requests, flag, array
\_of\_statuses, ierror) BIND(C) \fargs INTEGER, INTENT(IN) :: count \\
TYPE(MPI\_Request), INTENT(INOUT) :: array\_of\_requests(count) \\
LOGICAL, INTENT(OUT) :: flag \\ TYPE(MPI\_Status) :: array\_of
\_statuses(*) \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Testall(count, array\_of\_requests, flag, array
\_of\_statuses, ierror) \fargs INTEGER, INTENT(IN) :: count \\
TYPE(MPI\_Request), INTENT(INOUT) :: array\_of\_requests(count) \\
LOGICAL, INTENT(OUT) :: flag \\ TYPE(MPI\_Status) :: array\_of
\_statuses(*) \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
2676c2679
< \mpifnewbind{MPI\_Waitsome(incount, array\_of\_requests, outcount,
array\_of\_indices, array\_of\_statuses, ierror) BIND(C) \fargs
INTEGER, INTENT(IN) :: incount \\ TYPE(MPI\_Request), INTENT(INOUT) ::
array\_of\_requests(incount) \\ INTEGER, INTENT(OUT) :: outcount,
array\_of\_indices(*) \\ TYPE(MPI\_Status) :: array\_of\_statuses(*) \
\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Waitsome(incount, array\_of\_requests, outcount,
array\_of\_indices, array\_of\_statuses, ierror) \fargs INTEGER,
INTENT(IN) :: incount \\ TYPE(MPI\_Request), INTENT(INOUT) :: array
\_of\_requests(incount) \\ INTEGER, INTENT(OUT) :: outcount, array\_of
\_indices(*) \\ TYPE(MPI\_Status) :: array\_of\_statuses(*) \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
2732c2735
< \mpifnewbind{MPI\_Testsome(incount, array\_of\_requests, outcount,
array\_of\_indices, array\_of\_statuses, ierror) BIND(C) \fargs
INTEGER, INTENT(IN) :: incount \\ TYPE(MPI\_Request), INTENT(INOUT) ::
array\_of\_requests(incount) \\ INTEGER, INTENT(OUT) :: outcount,
array\_of\_indices(*) \\ TYPE(MPI\_Status) :: array\_of\_statuses(*) \
\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Testsome(incount, array\_of\_requests, outcount,
array\_of\_indices, array\_of\_statuses, ierror) \fargs INTEGER,
INTENT(IN) :: incount \\ TYPE(MPI\_Request), INTENT(INOUT) :: array
\_of\_requests(incount) \\ INTEGER, INTENT(OUT) :: outcount, array\_of
\_indices(*) \\ TYPE(MPI\_Status) :: array\_of\_statuses(*) \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
2748c2751
< {\sf fairness} requirement:  If a request for a receive repeatedly
---
> \mpiterm{fairness} requirement:  If a request for a receive
repeatedly
2778c2781
< \begin{example} {\rm
---
> \begin{example}
2816c2819
< } \end{example}
---
> \end{example}
2819c2822
< \begin{example} {\rm
---
> \begin{example}
2862c2865
< } \end{example}
---
> \end{example}
2868d2870
< \status{Passed twice.}
2886c2888
< \mpifnewbind{MPI\_Request\_get\_status(request, flag, status,
ierror) BIND(C) \fargs TYPE(MPI\_Request), INTENT(IN) :: request \\
LOGICAL, INTENT(OUT) :: flag \\ TYPE(MPI\_Status) :: status \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Request\_get\_status(request, flag, status,
ierror) \fargs TYPE(MPI\_Request), INTENT(IN) :: request \\ LOGICAL,
INTENT(OUT) :: flag \\ TYPE(MPI\_Status) :: status \\ INTEGER,
OPTIONAL, INTENT(OUT) :: ierror}
2906,2907c2908,2909
< The \func{MPI\_PROBE}, \func{MPI\_IPROBE},
< \func{MPI\_MPROBE}, and \func{MPI\_IMPROBE}
---
> The \mpifunc{MPI\_PROBE}, \mpifunc{MPI\_IPROBE},
> \mpifunc{MPI\_MPROBE}, and \mpifunc{MPI\_IMPROBE}
2915c2917
< The \func{MPI\_CANCEL} operation allows pending communications to be
cancelled.
---
> The \mpifunc{MPI\_CANCEL} operation allows pending communications to
be cancelled.
2934c2936
< \mpifnewbind{MPI\_Iprobe(source, tag, comm, flag, status, ierror)
BIND(C) \fargs INTEGER, INTENT(IN) :: source, tag \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ LOGICAL, INTENT(OUT) :: flag \\ TYPE(MPI
\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Iprobe(source, tag, comm, flag, status, ierror)
\fargs INTEGER, INTENT(IN) :: source, tag \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ LOGICAL, INTENT(OUT) :: flag \\ TYPE(MPI
\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
2941c2943
< \func{MPI\_IPROBE(source, tag, comm, flag, status)}
---
> \mpifunc{MPI\_IPROBE(source, tag, comm, flag, status)}
2947c2949
< that would have been received by a call to \mpifunc{MPI\_RECV(...,
source, tag,
---
> that would have been received by a call to \mpifunc{MPI\_RECV($
\ldots$, source, tag,
2950c2952
< value that would have been returned by \func{MPI\_RECV()}.
---
> value that would have been returned by \mpifunc{MPI\_RECV()}.
2954c2956
< If \func{MPI\_IPROBE} returns \mpiarg{flag = true},
---
> If \mpifunc{MPI\_IPROBE} returns \mpiarg{flag = true},
2979c2981
< see Section~\ref{sec:pt2pt-nullproc} on page~\pageref{sec:pt2pt-
nullproc}.
---
> see \sectionref{sec:pt2pt-nullproc}.
2991c2993
< \mpifnewbind{MPI\_Probe(source, tag, comm, status, ierror) BIND(C)
\fargs INTEGER, INTENT(IN) :: source, tag \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Status) :: status \\ INTEGER,
OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Probe(source, tag, comm, status, ierror) \fargs
INTEGER, INTENT(IN) :: source, tag \\ TYPE(MPI\_Comm), INTENT(IN) ::
comm \\ TYPE(MPI\_Status) :: status \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
2998c3000
< \func{MPI\_PROBE} behaves like \func{MPI\_IPROBE} except that it is
a blocking
---
> \mpifunc{MPI\_PROBE} behaves like \mpifunc{MPI\_IPROBE} except that
it is a blocking
3010c3012
< \mpifunc{MPI\_IPROBE} will eventually return {\sf flag = true}
---
> \mpifunc{MPI\_IPROBE} will eventually return \mpicode{flag = true}
3015c3017
< \begin{example} {\rm
---
> \begin{example}
3050c3052
< } \end{example}
---
> \end{example}
3053c3055
< \begin{example} {\rm
---
> \begin{example}
3062d3063
<
3099c3100
< \mpifunc{MPI\_PROBE}.}
---
> \mpifunc{MPI\_PROBE}.
3104c3105
< In a multithreaded MPI program, \mpifunc{MPI\_PROBE} and
---
> In a multithreaded \MPI/ program, \mpifunc{MPI\_PROBE} and
3116c3117
< message that would have been received by a call to \mpifunc{MPI
\_RECV(...,
---
> message that would have been received by a call to \mpifunc{MPI
\_RECV($\ldots$,
3119,3120c3120,3121
< that this message has source {\sf s}, tag {\sf t} and communicator
< {\sf c}.  If the
---
> that this message has source \mpicode{s}, tag \mpicode{t} and
communicator
```

```
> \mpicode{c}.  If the
3123,3125c3124,3126
< will be the earliest pending message from source {\sf s}
< with communicator {\sf c} and any tag; in any case, the message
probed will be the
< earliest pending message from source {\sf s} with tag {\sf t} and
---
> will be the earliest pending message from source \mpicode{s}
> with communicator \mpicode{c} and any tag; in any case, the message
probed will be the
> earliest pending message from source \mpicode{s} with tag
\mpicode{t} and
3127c3128
< {\sf c} (this is
---
> \mpicode{c} (this is
3130c3131
< from source {\sf s} with tag {\sf t} and communicator {\sf c}, until
it is received.
---
> from source \mpicode{s} with tag \mpicode{t} and communicator
\mpicode{c}, until it is received.
3139c3140
< The function \func{MPI\_PROBE} checks for incoming messages without
---
> The function \mpifunc{MPI\_PROBE} checks for incoming messages
without
3141c3142
< threads of each MPI process, it can be hard to use this
functionality
---
> threads of each \MPI/ process, it can be hard to use this
functionality
3144,3145c3145,3146
< Like \func{MPI\_PROBE} and \func{MPI\_IPROBE}, the
< \func{MPI\_MPROBE} and \func{MPI\_IMPROBE} operations allow incoming
---
> Like \mpifunc{MPI\_PROBE} and \mpifunc{MPI\_IPROBE}, the
> \mpifunc{MPI\_MPROBE} and \mpifunc{MPI\_IMPROBE} operations allow
incoming
3147c3148
< \func{MPI\_MPROBE} and \func{MPI\_IMPROBE} provide a mechanism to
receive the specific
---
> \mpifunc{MPI\_MPROBE} and \mpifunc{MPI\_IMPROBE} provide a mechanism
to receive the specific
3167,3168c3168,3169
< \mpifnewbind{MPI\_Improbe(source, tag, comm, flag, message, status,
ierror) BIND(C) \fargs INTEGER, INTENT(IN) :: source, tag \\ TYPE(MPI
\_Comm), INTENT(IN) :: comm \\ INTEGER, INTENT(OUT) :: flag \\
TYPE(MPI\_Message), INTENT(OUT) :: message \\ TYPE(MPI\_Status) ::
status \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
< \mpifbind{MPI\_IMPROBE(SOURCE, TAG, COMM, FLAG, MESSAGE, STATUS,
IERROR)\fargs INTEGER SOURCE, TAG, COMM, FLAG, MESSAGE, STATUS(MPI
\_STATUS\_SIZE), IERROR}
---
> \mpifnewbind{MPI\_Improbe(source, tag, comm, flag, message, status,
ierror) \fargs INTEGER, INTENT(IN) :: source, tag \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ LOGICAL, INTENT(OUT) :: flag \\ TYPE(MPI
\_Message), INTENT(OUT) :: message \\ TYPE(MPI\_Status) :: status \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
> \mpifbind{MPI\_IMPROBE(SOURCE, TAG, COMM, FLAG, MESSAGE, STATUS,
IERROR)\fargs INTEGER SOURCE, TAG, COMM, MESSAGE, STATUS(MPI\_STATUS
\_SIZE), IERROR \\ LOGICAL FLAG}
3174c3175
< would have been received by a call to \mpifunc{MPI\_RECV(...,
source, tag,
---
> would have been received by a call to \mpifunc{MPI\_RECV($\ldots$,
source, tag,
3181c3182
< A matched receive (\func{MPI\_MRECV} or \func{MPI\_IMRECV}) executed
with the message handle will receive the
---
> A matched receive (\mpifunc{MPI\_MRECV} or \mpifunc{MPI\_IMRECV})
executed with the message handle will receive the
3228c3229
< \mpifnewbind{MPI\_Mprobe(source, tag, comm, message, status, ierror)
BIND(C) \fargs INTEGER, INTENT(IN) :: source, tag \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Message), INTENT(OUT) :: message \\
TYPE(MPI\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) ::
ierror}
---
> \mpifnewbind{MPI\_Mprobe(source, tag, comm, message, status, ierror)
\fargs INTEGER, INTENT(IN) :: source, tag \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Message), INTENT(OUT) :: message \\
TYPE(MPI\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) ::
ierror}
3236c3237
< The implementation of \func{MPI\_MPROBE} and \func{MPI\_IMPROBE}
needs
---
> The implementation of \mpifunc{MPI\_MPROBE} and \mpifunc{MPI
\_IMPROBE} needs
3238c3239
< \func{MPI\_PROBE} and \func{MPI\_IPROBE}.
---
> \mpifunc{MPI\_PROBE} and \mpifunc{MPI\_IPROBE}.
3260c3261
< \mpifnewbind{MPI\_Mrecv(buf, count, datatype, message, status,
ierror) BIND(C) \fargs TYPE(*), DIMENSION(..) :: buf \\ INTEGER,
INTENT(IN) :: count \\ TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\
TYPE(MPI\_Message), INTENT(INOUT) :: message \\ TYPE(MPI\_Status) ::
status \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Mrecv(buf, count, datatype, message, status,
ierror) \fargs TYPE(*), DIMENSION(..) :: buf \\ INTEGER, INTENT(IN) ::
count \\ TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI
\_Message), INTENT(INOUT) :: message \\ TYPE(MPI\_Status) :: status \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3283c3284
< If \func{MPI\_MRECV} is called with
---
> If \mpifunc{MPI\_MRECV} is called with
3306c3307
< \mpifnewbind{MPI\_Imrecv(buf, count, datatype, message, request,
ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf \\
INTEGER, INTENT(IN) :: count \\ TYPE(MPI\_Datatype), INTENT(IN) ::
datatype \\ TYPE(MPI\_Message), INTENT(INOUT) :: message \\ TYPE(MPI
\_Request), INTENT(OUT) :: request \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Imrecv(buf, count, datatype, message, request,
ierror) \fargs TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf \\ INTEGER,
INTENT(IN) :: count \\ TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\
TYPE(MPI\_Message), INTENT(INOUT) :: message \\ TYPE(MPI\_Request),
INTENT(OUT) :: request \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3346c3347
< \mpifnewbind{MPI\_Cancel(request, ierror) BIND(C) \fargs TYPE(MPI
\_Request), INTENT(IN) :: request \\ INTEGER, OPTIONAL, INTENT(OUT) ::
ierror}
---
> \mpifnewbind{MPI\_Cancel(request, ierror) \fargs TYPE(MPI\_Request),
INTENT(IN) :: request \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3352c3353
< A call to \func{MPI\_CANCEL} marks for cancellation a pending,
---
> A call to \mpifunc{MPI\_CANCEL} marks for cancellation a pending,
3356,3359c3357,3360
< It is still necessary to call \func{MPI\_REQUEST\_FREE},
< \func{MPI\_WAIT} or \func{MPI\_TEST} (or any of the derived
operations)
< with the cancelled request as argument after the call to \func{MPI
\_CANCEL}.
< If a communication is marked for cancellation, then a \func{MPI
\_WAIT}
---
> It is still necessary to call \mpifunc{MPI\_REQUEST\_FREE},
> \mpifunc{MPI\_WAIT} or \mpifunc{MPI\_TEST} (or any of the derived
operations)
```

```
> with the cancelled request as argument after the call to
\mpifunc{MPI\_CANCEL}.
> If a communication is marked for cancellation, then a \mpifunc{MPI
\_WAIT}
3361,3362c3362,3363
< the activities of other processes (i.e., \func{MPI\_WAIT} behaves as
a
< local function); similarly if \func{MPI\_TEST} is
---
> the activities of other processes (i.e., \mpifunc{MPI\_WAIT} behaves
as a
> local function); similarly if \mpifunc{MPI\_TEST} is
3413c3414
< \mpifnewbind{MPI\_Test\_cancelled(status, flag, ierror) BIND(C)
\fargs TYPE(MPI\_Status), INTENT(IN) :: status \\ LOGICAL,
INTENT(OUT) :: flag \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Test\_cancelled(status, flag, ierror) \fargs
TYPE(MPI\_Status), INTENT(IN) :: status \\ LOGICAL, INTENT(OUT) ::
flag \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3423c3424
< operation might be cancelled then one should call \func{MPI\_TEST
\_CANCELLED}
---
> operation might be cancelled then one should call \mpifunc{MPI\_TEST
\_CANCELLED}
3454c3455
< binding the list of communication arguments to a {\bf persistent}
communication
---
> binding the list of communication arguments to a
\mpiterm{persistent} communication
3489c3490
< \mpifnewbind{MPI\_Send\_init(buf, count, datatype, dest, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Send\_init(buf, count, datatype, dest, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3516c3517
< \mpifnewbind{MPI\_Bsend\_init(buf, count, datatype, dest, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Bsend\_init(buf, count, datatype, dest, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3541c3542
< \mpifnewbind{MPI\_Ssend\_init(buf, count, datatype, dest, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Ssend\_init(buf, count, datatype, dest, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3566c3567
< \mpifnewbind{MPI\_Rsend\_init(buf, count, datatype, dest, tag, comm,
request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Rsend\_init(buf, count, datatype, dest, tag, comm,
request, ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, dest, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3591c3592
< \mpifnewbind{MPI\_Recv\_init(buf, count, datatype, source, tag,
comm, request, ierror) BIND(C) \fargs TYPE(*), DIMENSION(..),
ASYNCHRONOUS :: buf \\ INTEGER, INTENT(IN) :: count, source, tag \\
TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Recv\_init(buf, count, datatype, source, tag,
comm, request, ierror) \fargs TYPE(*), DIMENSION(..), ASYNCHRONOUS ::
buf \\ INTEGER, INTENT(IN) :: count, source, tag \\ TYPE(MPI
\_Datatype), INTENT(IN) :: datatype \\ TYPE(MPI\_Comm), INTENT(IN) ::
```

```
comm \\ TYPE(MPI\_Request), INTENT(OUT) :: request \\ INTEGER,
OPTIONAL, INTENT(OUT) :: ierror}
3598c3599
< argument to \func{MPI\_RECV\_INIT}.
---
> argument to \mpifunc{MPI\_RECV\_INIT}.
3613c3614
< \mpifnewbind{MPI\_Start(request, ierror) BIND(C) \fargs TYPE(MPI
\_Request), INTENT(INOUT) :: request \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Start(request, ierror) \fargs TYPE(MPI\_Request),
INTENT(INOUT) :: request \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
3653c3654
< \mpifnewbind{MPI\_Startall(count, array\_of\_requests, ierror)
BIND(C) \fargs INTEGER, INTENT(IN) :: count \\ TYPE(MPI\_Request),
INTENT(INOUT) :: array\_of\_requests(count) \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Startall(count, array\_of\_requests, ierror)
\fargs INTEGER, INTENT(IN) :: count \\ TYPE(MPI\_Request),
INTENT(INOUT) :: array\_of\_requests(count) \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
3662c3663
< \func{MPI\_STARTALL(count, array\_of\_requests)} has the
---
> \mpifunc{MPI\_STARTALL(count, array\_of\_requests)} has the
3664,3665c3665,3666
< \func{MPI\_START} \mpiarg{(\&array\_of\_requests[i])},
< executed for {\sf i=0 ,..., count-1}, in some arbitrary order.
---
> \mpifunc{MPI\_START} \mpiarg{(\&array\_of\_requests[i])},
> executed for \mpicode{i=0 ,$\ldots$, count-1}, in some arbitrary
order.
3689,3690c3690,3691
< \( \bf
< Create \ (Start \ Complete)^* \ Free
---
> \(
> \textbf{Create \ (Start \ Complete)$^*$ \ Free}
3698c3699
< A send operation initiated with \func{MPI\_START} can be matched
with
---
> A send operation initiated with \mpifunc{MPI\_START} can be matched
with
3700c3701
< with \func{MPI\_START} can receive messages generated by any send
---
> with \mpifunc{MPI\_START} can receive messages generated by any send
```

```
3702a3704,3710
> %% Note that a previous version of this included what it hoped were
the
> %% titles of the sections.  That hope was misplaced, as the section
> %% titles had been changed.  Another example of the folly in trying
to
> %% keep multiple references manually synchronized.  The titles are
really
> %% not necessary here, as the first sentance covers the general
issues.
> %% Attempting to enumerate them all is also a mistake, as it leaves
out
> %% some that are also important.
3706,3715c3714,3723
< Sections~\ref{sec:misc-problems}-\ref{sec:f90-problems:comparison-
with-C},
< especially in
< Sections~\ref{sec:misc-sequence} and~\ref{sec:f90-problems:vector-
subscripts}
< on pages~\pageref{sec:misc-sequence}-\pageref{sec:f90-
problems:vector-subscripts}
< about ``{\sf Problems Due to Data Copying and Sequence Association
with Subscript Triplets}''
< and ``{\sf Vector Subscripts}'',
< and in Sections~\ref{sec:misc-register} to~\ref{sec:f90-
problems:perm-data-movements}
< on pages~\pageref{sec:misc-register} to~\pageref{sec:f90-
problems:perm-data-movements}
< about ``{\sf Optimization Problems}'', ``{\sf Code Movements and
Register Optimization}'',
< ``{\sf Temporary Data Movements}'' and ``{\sf Permanent Data
Movements}''.
---
> Sections~\ref{sec:misc-problems}--\ref{sec:f90-problems:comparison-
with-C}.
> %% especially in
> %% Sections~\ref{sec:misc-sequence} and~\ref{sec:f90-
problems:vector-subscripts}
> %% on pages~\pageref{sec:misc-sequence}-\pageref{sec:f90-
problems:vector-subscripts}
> %% about ``{\sf Problems Due to Data Copying and Sequence
Association with Subscript Triplets}''
> %% and ``{\sf Vector Subscripts}'',
> %% and in Sections~\ref{sec:misc-register} to~\ref{sec:f90-
problems:perm-data-movements}
> %% on pages~\pageref{sec:misc-register} to~\pageref{sec:f90-
problems:perm-data-movements}
> %% about ``{\sf Optimization Problems}'', ``{\sf Code Movements and
Register Optimization}'',
> %% ``{\sf Temporary Data Movements}'' and ``{\sf Permanent Data
Movements}''.
3722c3730
< The {\bf send-receive} operations combine in one call the sending of
a
---
> The \mpiterm{send-receive} operations combine in one call the
sending of a
3769c3777
< \mpifnewbind{MPI\_Sendrecv(sendbuf, sendcount, sendtype, dest,
sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status,
ierror) BIND(C) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: sendbuf \
\ TYPE(*), DIMENSION(..) :: recvbuf \\ INTEGER, INTENT(IN) ::
sendcount, dest, sendtag, recvcount, source, recvtag \\ TYPE(MPI
\_Datatype), INTENT(IN) :: sendtype, recvtype \\ TYPE(MPI\_Comm),
INTENT(IN) :: comm \\ TYPE(MPI\_Status) :: status \\ INTEGER,
OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Sendrecv(sendbuf, sendcount, sendtype, dest,
sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status,
ierror) \fargs TYPE(*), DIMENSION(..), INTENT(IN) :: sendbuf \\
TYPE(*), DIMENSION(..) :: recvbuf \\ INTEGER, INTENT(IN) :: sendcount,
dest, sendtag, recvcount, source, recvtag \\ TYPE(MPI\_Datatype),
INTENT(IN) :: sendtype, recvtype \\ TYPE(MPI\_Comm), INTENT(IN) ::
comm \\ TYPE(MPI\_Status) :: status \\ INTEGER, OPTIONAL,
INTENT(OUT) :: ierror}
3804c3812
< \mpifnewbind{MPI\_Sendrecv\_replace(buf, count, datatype, dest,
sendtag, source, recvtag, comm, status, ierror) BIND(C) \fargs
TYPE(*), DIMENSION(..) :: buf \\ INTEGER, INTENT(IN) :: count, dest,
sendtag, source, recvtag \\ TYPE(MPI\_Datatype), INTENT(IN) ::
datatype \\ TYPE(MPI\_Comm), INTENT(IN) :: comm \\ TYPE(MPI
\_Status) :: status \\ INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
---
> \mpifnewbind{MPI\_Sendrecv\_replace(buf, count, datatype, dest,
sendtag, source, recvtag, comm, status, ierror) \fargs TYPE(*),
DIMENSION(..) :: buf \\ INTEGER, INTENT(IN) :: count, dest, sendtag,
source, recvtag \\ TYPE(MPI\_Datatype), INTENT(IN) :: datatype \\
TYPE(MPI\_Comm), INTENT(IN) :: comm \\ TYPE(MPI\_Status) :: status \\
INTEGER, OPTIONAL, INTENT(OUT) :: ierror}
```