MPI_SENDRECV(sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status)

IN	sendbuf	initial address of send buffer (choice)	
IN	sendcount	number of elements in send buffer (non-negative integer)	
IN	sendtype	type of elements in send buffer (handle)	
IN	dest	rank of destination (integer)	
IN	sendtag	send tag (integer)	
OUT	recvbuf	initial address of receive buffer (choice)	
IN	recvcount	number of elements in receive buffer (non-negative integer)	
IN	recvtype	type of elements in receive buffer (handle)	
IN	source	rank of source or MPI_ANY_SOURCE (integer)	ticket51.
IN	recvtag	receive tag or MPI_ANY_TAG (integer)	ticket51.
IN	comm	communicator (handle)	
OUT	status	status object (Status)	

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

```
MPI_SENDRECV(SENDBUF, SENDCOUNT, SENDTYPE, DEST, SENDTAG, RECVBUF,
             RECVCOUNT, RECVTYPE, SOURCE, RECVTAG, COMM, STATUS, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNT, SENDTYPE, DEST, SENDTAG, RECVCOUNT, RECVTYPE,
    SOURCE, RECVTAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR
```

ticket150.

```
{void MPI::Comm::Sendrecv(const void *sendbuf, int sendcount, const
             MPI::Datatype& sendtype, int dest, int sendtag, void *recvbuf,
             int recvcount, const MPI::Datatype& recvtype, int source,
             int recvtag, MPI::Status& status) const (binding deprecated, see
             Section 15.2) }
```

ticket150.

ticket150.

```
{void MPI::Comm::Sendrecv(const void *sendbuf, int sendcount, const
             MPI::Datatype& sendtype, int dest, int sendtag, void *recvbuf,
             int recvcount, const MPI::Datatype& recvtype, int source,
             int recvtag) const (binding deprecated, see Section 15.2) }
```

ticket150.

Execute a blocking send and receive operation. Both send and receive use the same communicator, but possibly different tags. The send buffer and receive buffers must be disjoint, and may have different lengths and datatypes.

The semantics of a send-receive operation is what would be obtained if the caller forked two concurrent threads, one to execute the send, and one to execute the receive, followed by a join of these two threads.

MPI_SENDRECV_REPLACE(buf, count, datatype, dest, sendtag, source, recvtag, comm, status)

INOUT	buf	initial address of send and receive buffer (choice)
IN	count	number of elements in send and receive buffer (non-negative integer)
IN	datatype	type of elements in send and receive buffer (handle)
IN	dest	rank of destination (integer)
ticket51. IN	sendtag	send message tag or MPI_ANY_TAG (integer)
ticket51. IN	source	rank of source or MPI_ANY_SOURCE (integer)
IN	recvtag	receive message tag (integer)
IN	comm	communicator (handle)
OUT	status	status object (Status)

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

```
MPI_SENDRECV_REPLACE(BUF, COUNT, DATATYPE, DEST, SENDTAG, SOURCE, RECVTAG,
             COMM, STATUS, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, SENDTAG, SOURCE, RECVTAG, COMM,
    STATUS(MPI_STATUS_SIZE), IERROR
```

ticket150.

```
{void MPI::Comm::Sendrecv_replace(void* buf, int count, const
             MPI::Datatype& datatype, int dest, int sendtag, int source,
             int recvtag, MPI::Status& status) const (binding deprecated, see
             Section 15.2) }
```

ticket150.

ticket150.

```
{void MPI::Comm::Sendrecv_replace(void* buf, int count, const
             MPI::Datatype& datatype, int dest, int sendtag, int source,
             int recvtag) const (binding deprecated, see Section 15.2) }
```

ticket150.

Execute a blocking send and receive. The same buffer is used both for the send and for the receive, so that the message sent is replaced by the message received.

> *Advice to implementors.* Additional intermediate buffering is needed for the "replace" variant. (*End of advice to implementors.*)

3.11 Null Processes

In many instances, it is convenient to specify a "dummy" source or destination for communication. This simplifies the code that is needed for dealing with boundaries, for example, in the case of a non-circular shift done with calls to send-receive.

The special value MPI_PROC_NULL can be used instead of a rank wherever a source or a destination argument is required in a call. A communication with process MPI_PROC_NULL has no effect. A send to MPI_PROC_NULL succeeds and returns as soon as possible. A receive