comm argument. This will allow communication with all the processes available at initialization time.

Users may define new communicators, as explained in Chapter 6. Communicators provide an important encapsulation mechanism for libraries and modules. They allow modules to have their own disjoint communication universe and their own process numbering scheme. (*End of advice to users.*)

Advice to implementors. The message envelope would normally be encoded by a fixed-length message header. However, the actual encoding is implementation dependent. Some of the information (e.g., source or destination) may be implicit, and need not be explicitly carried by messages. Also, processes may be identified by relative ranks, or absolute ids, etc. (*End of advice to implementors.*)

3.2.4 Blocking Receive

The syntax of the blocking receive operation is given below.

MPI_RECV (buf, count, datatype, source, tag, comm, status)

	OUT	buf	initial address of receive buffer (choice)
	IN	count	number of elements in receive buffer (non-negative integer)
	IN	datatype	datatype of each receive buffer element (handle)
ticket51.	IN	source	rank of source or MPI_ANY_SOURCE (integer)
ticket51.	IN	tag	message tag or MPI_ANY_TAG (integer)
	IN	comm	communicator (handle)
	OUT	status	status object (Status)

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

```
MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE),
    IERROR
```

ticket150.
```
{void MPI::Comm::Recv(void* buf, int count, const MPI::Datatype& datatype,
             int source, int tag, MPI::Status& status) const (binding
             deprecated, see Section 15.2) }
```
ticket150.

ticket150.
```
{void MPI::Comm::Recv(void* buf, int count, const MPI::Datatype& datatype,
             int source, int tag) const (binding deprecated, see Section 15.2) }
```
ticket150.

The blocking semantics of this call are described in Section 3.4.

The receive buffer consists of the storage containing count consecutive elements of the type specified by datatype, starting at address buf. The length of the received message must be less than or equal to the length of the receive buffer. An overflow error occurs if all incoming data does not fit, without truncation, into the receive buffer.