

Advice to implementors. A call to `MPI_PROBE(source, tag, comm, status)` will match the message that would have been received by a call to `MPI_RECV(..., source, tag, comm, status)` executed at the same point. Suppose that this message has source `s`, tag `t` and communicator `c`. If the tag argument in the probe call has value `MPI_ANY_TAG` then the message probed will be the earliest pending message from source `s` with communicator `c` and any tag; in any case, the message probed will be the earliest pending message from source `s` with tag `t` and communicator `c` (this is the message that would have been received, so as to preserve message order). This message continues as the earliest pending message from source `s` with tag `t` and communicator `c`, until it is received. A receive operation subsequent to the probe that uses the same communicator as the probe and uses the tag and source values returned by the probe, must receive this message, unless it has already been received by another receive operation. (*End of advice to implementors.*)

`MPI_CANCEL(request)`

IN request communication request (handle)

`int MPI_Cancel(MPI_Request *request)`

`MPI_CANCEL(REQUEST, IERROR)`

INTEGER REQUEST, IERROR

`{void MPI::Request::Cancel() const(binding deprecated, see Section 15.2) }`

A call to `MPI_CANCEL` marks for cancellation a pending, nonblocking communication operation (send or receive). The cancel call is local. It returns immediately, possibly before the communication is actually canceled. [It is still necessary to complete a communication that has been marked for cancellation,] It is still necessary to free the request of the communication that was marked for cancellation, using a call to `MPI_REQUEST_FREE`, `MPI_WAIT` or `MPI_TEST` (or any of the derived operations).

If a communication is marked for cancellation, then a `MPI_WAIT` call for that communication is guaranteed to return, irrespective of the activities of other processes (i.e., `MPI_WAIT` behaves as a local function); similarly if `MPI_TEST` is repeatedly called in a busy wait loop for a canceled communication, then `MPI_TEST` will eventually be successful.

`MPI_CANCEL` can be used to cancel a communication that uses a persistent request (see Section 3.9), in the same way it is used for nonpersistent requests. A successful cancellation cancels the active communication, but not the request itself. After the call to `MPI_CANCEL` and the subsequent call to `MPI_WAIT` or `MPI_TEST`, the request becomes inactive and can be activated for a new communication.

The successful cancellation of a buffered send frees the buffer space occupied by the pending message.

Either the cancellation succeeds, or the communication succeeds, but not both. If a send is marked for cancellation, then it must be the case that either the send completes normally, in which case the message sent was received at the destination process, or that the send is successfully canceled, in which case no part of the message was received at the destination. Then, any matching receive has to be satisfied by another send. If a receive is