ticket150.

{void MPI::Intracomm::Scan(const void* sendbuf, void* recvbuf, int count,
const MPI::Datatype& datatype, const MPI::Op& op) const
(binding deprecated, see Section ??) } ticket150.

If comm is an intracommunicator, MPI_SCAN is used to perform a prefix reduction on data distributed across the group. The operation returns, in the receive buffer of the process with rank i, the reduction of the values in the send buffers of processes with ranks 0,...,i (inclusive). The type of operations supported, their semantics, and the constraints on send and receive buffers are as for MPI_REDUCE.

The "in place" option for intracommunicators is specified by passing MPI_IN_PLACE in the sendbuf argument. In this case, the input data is taken from the receive buffer, and replaced by the output data.

This operation is invalid for intercommunicators.

5.11.2 Exclusive Scan

MPI_EXSCAN(sendbuf, recvbuf, count, datatype, op, comm)

IN	sendbuf	starting address of send buffer (choice)
OUT	recvbuf	starting address of receive buffer (choice)
IN	count	number of elements in input buffer (non-negative integer)
IN	datatype	data type of elements of input buffer (handle)
IN	op	operation (handle)
IN	comm	intracommunicator (handle)

```
int MPI_Exscan(void *sendbuf, void *recvbuf, int count,
              MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

MPI_EXSCAN(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER COUNT, DATATYPE, OP, COMM, IERROR
```

ticket150.

{void MPI::Intracomm::Exscan(const void* sendbuf, void* recvbuf, int count,
const MPI::Datatype& datatype, const MPI::Op& op) const
(binding deprecated, see Section ??) } ticket150.

If comm is an intracommunicator, MPI_EXSCAN is used to perform a prefix reduction on data distributed across the group. The value in recvbuf on the process with rank 0 is undefined, and recvbuf is not signficant on process 0. The value in recvbuf on the process with rank 1 is defined as the value in sendbuf on the process with rank 0. For processes with rank $i > 1$, the operation returns, in the receive buffer of the process with rank i, the reduction of the values in the send buffers of processes with ranks $0, \ldots, i-1$ (inclusive). The type of operations supported, their semantics, and the constraints on send and receive buffers, are as for MPI_REDUCE.

ticket94.

[No "in place" option is supported.]The "in place" option for intracommunicators is specified by passing MPI_IN_PLACE in the sendbuf argument. In this case, the input data is taken from the receive buffer, and replaced by the output data. The receive buffer on rank 0 is not changed by this operation.

ticket92. This operation is invalid for intercommunicators.
[MPI-2.2 - ticket #92, passed Jun 8-10, 2009

Advice to users. As for MPI_SCAN, MPI does not specify which processes may call the operation, only that the result be correctly computed. In particular, note that the process with rank 1 need not call the MPI_Op, since all it needs to do is to receive the value from the process with rank 0. However, all processes, even the processes with ranks zero and one, must provide the same op. (*End of advice to users.*)

]

Rationale. The exclusive scan is more general than the inclusive scan. Any inclusive scan operation can be achieved by using the exclusive scan and then locally combining the local contribution. Note that for non-invertable operations such as MPI_MAX, the

ticket94. exclusive scan cannot be computed with the inclusive scan.
[MPI-2.2 - ticket #94, passed Jul 27-29, 2009 MPI-2.1 Ballots 1-4 No in-place version is specified for MPI_EXSCAN because it is not clear what this means for the process with rank zero. MPI-2.1 Ballots 1-4] (*End of rationale.*)

5.11.3 Example using MPI_SCAN

The example in this section uses an intracommunicator.

Example 5.22 This example uses a user-defined operation to produce a *segmented scan*. A segmented scan takes, as input, a set of values and a set of logicals, and the logicals delineate the various segments of the scan. For example:

values	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
logicals	0	0	1	1	1	0	0	1
result	v_1	$v_1 + v_2$	v_3	$v_3 + v_4$	$v_3 + v_4 + v_5$	v_6	$v_6 + v_7$	v_8

The operator that produces this effect is,

$$\begin{pmatrix} u \\ i \end{pmatrix} \circ \begin{pmatrix} v \\ j \end{pmatrix} = \begin{pmatrix} w \\ j \end{pmatrix},$$

where,

$$w = \begin{cases} u + v & \text{if } i = j \\ v & \text{if } i \neq j \end{cases}.$$

Note that this is a non-commutative operator. C code that implements it is given below.