Chapter 1

Errata for MPI-2

This document was processed on October 1, 2007.

The known corrections to MPI-2 are listed in this document. All page and line numbers are for the official version of the MPI-2 document available from the MPI Forum home page at `http://www.mpi-forum.org`. Information on reporting mistakes in the MPI documents is also located on the MPI Forum home page.

- Page 24, lines 20-21 read
  MPI_FINALIZE is collective on MPI_COMM_WORLD.
  but should read
  MPI_FINALIZE is collective over all connected processes. If no processes were spawned, accepted or connected then this means over MPI_COMM_WORLD; otherwise it is collective over the union of all processes that have been and continue to be connected, as explained in Section 5.5.4.

- Page 27, line 26 reads
  must be added to line 3 of page 54.
  but should read
  must be added to line 3 of page 52.

- Add to page 36, after line 3

  3.2.11 MPI_GET_COUNT with zero-length datatypes

  The value returned as the `count` argument of `MPI_GET_COUNT` for a datatype of length zero where zero bytes have been transferred is zero. If the number of bytes transfered is greater than zero, `MPI_UNDEFINED` is returned.

  > *Rationale.* Zero-length datatypes may be created in a number of cases. In MPI-2, an important case is `MPI_TYPE_CREATE_DARRAY`, where the definition of the particular darry results in an empty block on some MPI process. Programs written in an SPMD style will not check for this special case and may want to use `MPI_GET_COUNT` to check the status. (*End of rationale.*)

- Add to page 36, after 3.2.11 (above)

  3.2.12 MPI_GROUP_TRANSLATE_RANKS and MPI_PROC_NULL

MPI_PROC_NULL is a valid rank for input to MPI_GROUP_TRANSLATE_RANKS, which returns MPI_PROC_NULL as the translated rank.

- Page 51, after line 43, add

  ```
  MPI_Errhandler MPI_Errhandler_f2c(MPI_Fint errhandler)

  MPI_Fint MPI_Errhandler_c2f(MPI_Errhandler errhandler)
  ```

  These were overlooked.

- Page 53, line 7 reads

  ```
  void cpp_lib_call(MPI::Comm& cpp_comm);
  ```

  but should read

  ```
  void cpp_lib_call(MPI::Comm cpp_comm);
  ```

- Page 60, Line 1 reads

  ```
  char name[MPI_MAX_NAME_STRING];
  ```

  but should read

  ```
  char name[MPI_MAX_OBJECT_NAME];
  ```

  since MPI_MAX_NAME_STRING is not an MPI-defined constant.

- Page 61, after line 36. Add the following (paralleling the errata to MPI-1.1):

  MPI_{COMM,WIN,FILE}_GET_ERRHANDLER behave as if a new error handler object is created. That is, once the error handler is no longer needed, MPI_ERRHANDLER_FREE should be called with the error handler returned from MPI_ERRHANDLER_GET or MPI_{COMM,WIN,FILE}_GET_ to mark the error handler for deallocation. This provides behavior similar to that of MPI_COMM_GROUP and MPI_GROUP_FREE.

- Page 69, lines 14-15 read

  ```
  MPI::Datatype MPI::Datatype::Resized(const MPI::Aint lb,
  const MPI::Aint extent) const
  ```

  but should read

  ```
  MPI::Datatype MPI::Datatype::Create_resized(const MPI::Aint lb,
  const MPI::Aint extent) const
  ```

- On Page 78, after line 27, add:

  MPI_BYTE should be used to send and receive data that is packed using MPI_PACK_EXTERNAL.

> *Rationale.* MPI_PACK_EXTERNAL specifies that there is no header on the message and further specifies the exact format of the data. Since MPI_PACK may (and is allowed to) use a header, the datatype MPI_PACKED cannot be used for data packed with MPI_PACK_EXTERNAL. (*End of rationale.*)

- On page 93 after line 48, add

  Many of the descriptions of the collective routines provide illustrations in terms of blocking MPI point-to-point routines. These are intended solely to indicate what data is sent or received by what process. Many of these examples are *not* correct MPI programs; for purposes of simplicity, they often assume infinite buffering.

- Page 94, line 29 reads
  are the original sets of of processes.
  but should read
  are the original sets of processes.

- Page 114, after line 4, add

  MPI_PROC_NULL is a valid target rank in the MPI RMA calls MPI_ACCUMULATE, MPI_GET, and MPI_PUT. The effect is the same as for MPI_PROC_NULL in MPI point-to-point communication.

- Page 116, line 31, reads

```
void MPI::Win::Get(const void *origin_addr, int origin_count, const
MPI::Datatype& origin_datatype, int target_rank, MPI::Aint target_disp,
int target_count, const MPI::Datatype& target_datatype) const
```

  but should read

```
void MPI::Win::Get(void *origin_addr, int origin_count, const
MPI::Datatype& origin_datatype, int target_rank, MPI::Aint target_disp,
int target_count, const MPI::Datatype& target_datatype) const
```

- Page 120, after line 13:

  MPI_REPLACE, like the other predefined operations, is defined only for the predefined MPI datatypes.

  > *Rationale.* The rationale for this is that, for consistency, MPI_REPLACE should have the same limitations as the other operations. Extending it to all datatypes doesn't provide any real benefit. (*End of rationale.*)

- Page 162, lines 43–44 curently read
  The "in place" option for intracommunicators is specified by passing the value MPI_IN_PLACE to the argument sendbuf at the root.
  but should read
  The "in place" option for intracommunicators is specified by passing the value MPI_IN_PLACE to the argument sendbuf at all processes.

- Page 162, line 48 reads

  Both groups should provide the same count value.

but should read

Both groups should provide count and datatype arguments that specify the same type signature.

- Page 165, lines 4–22 read

IN	sendcounts	integer array equal to the group size specifying the number of elements to send to each processor (integer)
IN	sdispls	integer array (of length group size). Entry j specifies the displacement in bytes (relative to sendbuf) from which to take the outgoing data destined for process j
IN	sendtypes	array of datatypes (of length group size). Entry j specifies the type of data to send to process j (handle)
OUT	recvbuf	address of receive buffer (choice)
IN	recvcounts	integer array equal to the group size specifying the number of elements that can be received from each processor (integer)
IN	rdispls	integer array (of length group size). Entry i specifies the displacement in bytes (relative to recvbuf) at which to place the incoming data from process i
IN	recvtypes	array of datatypes (of length group size). Entry i specifies the type of data received from process i (handle)

but should read

IN	sendcounts	integer array equal to the group size specifying the number of elements to send to each processor (array of integers)
IN	sdispls	integer array (of length group size). Entry j specifies the displacement in bytes (relative to sendbuf) from which to take the outgoing data destined for process j (array of integers)
IN	sendtypes	array of datatypes (of length group size). Entry j specifies the type of data to send to process j (array of handles)
OUT	recvbuf	address of receive buffer (choice)
IN	recvcounts	integer array equal to the group size specifying the number of elements that can be received from each processor (array of integers)
IN	rdispls	integer array (of length group size). Entry i specifies the displacement in bytes (relative to recvbuf) at which to place the incoming data from process i (array of integers)
IN	recvtypes	array of datatypes (of length group size). Entry i specifies the type of data received from process i (array of handles)

- Page 199, after line 11, add:

  Advice to implementors. High quality implementations should raise an error when a keyval that was created by a call to `MPI_XXX_CREATE_KEYVAL` is used with an object of the wrong type with a call to `MPI_YYY_GET_ATTR`, `MPI_YYY_SET_ATTR`, `MPI_YYY_DELETE_ATTR`, or `MPI_YYY_FREE_KEYVAL`. To do so, it is necessary to maintain, with each keyval, information on the type of the associated user function. (*End of advice to implementors.*)

- Page 204, line 30 reads

```
bool MPI::Win::Get_attr(const MPI::Win& win, int win_keyval,
void* attribute_val) const
```

but should read

```
bool MPI::Win::Get_attr(int win_keyval, void* attribute_val) const
```

- Page 221, after line 40, add

`MPI_DISPLACEMENT_CURRENT` is invalid unless the amode for the file has `MPI_MODE_SEQUENTIAL` set.

- Page 230, line 17, reads

If `MPI_MODE_SEQUENTIAL` mode was specified when the file was opened, it is erroneous to call the routines in this section.

but should read

If MPI_MODE_SEQUENTIAL mode was specified when the file was opened, it is erroneous to call the routines in this section, with the exception of MPI_FILE_GET_BYTE_OFFSET.

- Page 250, line 8 reads
  with 15 exponent bits, bias = +10383, 112 fraction bits,
  but should read
  with 15 exponent bits, bias = +16383, 112 fraction bits,

- Page 251, Line 18 reads

```
        MPI_LONG_LONG             8
```

  but should read

```
        MPI_LONG_LONG_INT         8
```

  In addition, the type MPI_LONG_LONG should be added as an optional type; it is a synonym for MPI_LONG_LONG_INT.

- Page 253, line 4 reads

```
typedef MPI::Datarep_extent_function(const MPI::Datatype& datatype,
```

  but should read

```
typedef void MPI::Datarep_extent_function(const MPI::Datatype& datatype,
```

- Page 253, lines 22-24 read

```
typedef MPI::Datarep_conversion_function(void* userbuf,
MPI::Datatype& datatype, int count, void* filebuf,
MPI::Offset position, void* extra_state);
```

  but should read

```
typedef void MPI::Datarep_conversion_function(void* userbuf,
MPI::Datatype& datatype, int count, void* filebuf,
MPI::Offset position, void* extra_state);
```

- Page 273, line 24 reads

```
  void Send(void* buf, int count, const MPI::Datatype& type,
```

  but should read

```
  void Send(const void* buf, int count, const MPI::Datatype& type,
```

- Page 332, lines 23-24 read
  `MPI::Datatype MPI::Datatype::Resized(const MPI::Aint lb,`
  `const MPI::Aint extent) const`

  but should read
  `MPI::Datatype MPI::Datatype::Create_resized(const MPI::Aint lb,`
  `const MPI::Aint extent) const`

- Page 334, line 22 read

  ```
  void MPI::Win::Get(const void *origin_addr, int origin_count, const
  ```

  but should read

  ```
  void MPI::Win::Get(void *origin_addr, int origin_count, const
  ```

- Page 341, line 18 reads

  ```
  typedef MPI::Datarep_conversion_function(void* userbuf,
  ```

  but should read

  ```
  typedef void MPI::Datarep_conversion_function(void* userbuf,
  ```

- Page 341, line 22 reads

  ```
  typedef MPI::Datarep_extent_function(const MPI::Datatype& Datatype,
  ```

  but should read

  ```
  typedef void MPI::Datarep_extent_function(const MPI::Datatype& Datatype,
  ```

- Page 343, line 44
  Remove the `const` from `const MPI::Datatype`.

- Page 344, lines 13, 23, 32, 38, and 47
  Remove the `const` from `const MPI::Datatype`.

- Page 345, lines 5 and 11
  Remove the `const` from `const MPI::Datatype`.

- Page 346, line 16 reads

  ```
  // Type: MPI::Errhandler
  ```

  but should read

  ```
  // Type: const MPI::Errhandler
  ```

- Page 354, line 17 reads

```
void Get_version(int& version, int& subversion);
```

  but should read

```
void Get_version(int& version, int& subversion)
```

- Page 354, lines 25-30 read

```
Exception::Exception(int error_code);

int Exception::Get_error_code() const;

int Exception::Get_error_class() const;

const char* Exception::Get_error_string() const;
```

  but should read

```
Exception::Exception(int error_code)

int Exception::Get_error_code() const

int Exception::Get_error_class() const

const char* Exception::Get_error_string() const
```

- Page 357, line 24 reads
  MPI_CART_RANK `Cartcomm` `Get_rank` `int rank`
  but should read
  MPI_CART_RANK `Cartcomm` `Get_cart_rank` `int rank`

- Page 359, line 27 reads
  MPI_TOPO_TEST `Comm` `Get_topo` `int status`
  but should read
  MPI_TOPO_TEST `Comm` `Get_topology` `int status`

- MPI_COMM_PARENT instead of MPI_COMM_GET_PARENT **(Pending Errata (see Ballot 3))**

  Page 179, lines 4-5 change

  Thus, the names of MPI_COMM_WORLD, MPI_COMM_SELF, and MPI_COMM_PARENT will have the default of MPI_COMM_WORLD, MPI_COMM_SELF, and MPI_COMM_PARENT.

  to

  Thus, the names of MPI_COMM_WORLD, MPI_COMM_SELF, and the communicator returned by MPI_COMM_GET_PARENT (if not MPI_COMM_NULL) will have the default of MPI_COMM_WORLD, MPI_COMM_SELF, and MPI_COMM_PARENT.

Page 94, line 3-5, change

```
* The manager is represented as the process with rank 0 in (the remote
* group of) MPI\_COMM\_PARENT. If the workers need to communicate among
* themselves, they can use MPI\_COMM\_WORLD.
```

to

```
* The manager is represented as the process with rank 0 in (the remote
* group of) the parent communicator. If the workers need to communicate
* among themselves, they can use MPI\_COMM\_WORLD.
```

- MPI_UNPACK_EXTERNAL **(Pending Errata (see Ballot 3))**

  Page 79, Line 11 is

  ```
  MPI_UNPACK_EXTERNAL (datarep, inbuf, incount, datatype, outbuf, outsize,
  position)
  ```

  but should be

  ```
  MPI_UNPACK_EXTERNAL (datarep, inbuf, insize, position, outbuf, outcount,
  datatype)
  ```

- Additional C++ binding errors **(Pending Errata (see Ballot 3))**

  Page 337, line 31-32 reads

  ```
  bool MPI::Win::Get_attr(const MPI::Win&win, int win_keyval,
                void* attribute_val) const
  ```

  but should read

  ```
  bool MPI::Win::Get_attr(int win_keyval, void* attribute_val) const
  ```

- MPI_REQUEST_CANCEL used where MPI_CANCEL intended **(Pending Errata (see Ballot 3))**

  On page 172 in section 8.2, change `MPI_REQUEST_CANCEL` to `MPI_CANCEL`.

- Intercommunicator collective and datatypes **(Pending Errata (see Ballot 3))**

  Page 163, line 22 reads

  > Within each group, all processes provide the same `recvcounts` argument, and the sum of the `recvcounts` entries should be the same for the two groups.

  but should read

Within each group, all processes provide the same `recvcounts` argument, and the sum of the `recvcounts` entries and `datatype` should specify the same type signature for the two groups.

- `const` in C++ specification of predefined MPI objects **(Pending Errata (see Ballot 3))**

  Page 345, line 37: Remove the const from const MPI::Op.

  Page 346, line 20: Remove the const from const MPI::Group.

  Page 346, add after line 34:

  > *Advice to implementors.* If an implementation does not change the value of predefined handles while execution of MPI_Init, the implementation is free to define the predefined operation handles as const MPI::Op and the predefined group handle MPI::GROUP_EMPTY as const MPI::Group. Other predefined handles must not be "const" because they are allowed as INOUT argument in the MPI_COMM_SET_NAME/ATTR and MPI_TYPE_SET_NAME/ATTR routines. (*End of advice to implementors.*)