Instructions for Preparing the MPI Standard Document

Message Passing Interface Forum

October 9, 2010

1 Introduction

This document provides guidance on editing the MPI standard documents. The MPI standard uses LaTeX, a powerful markup language where items are marked based on the content, rather than low-level control of individual formatting options as in WYSIWYG (what you see is what you get) markup. LaTeX is a high level interface over TeX, a powerful and general purpose typesetting language. LaTeX permits the use of TeX, which gives it extra power but also makes it easy to introduce the sort of inconsistent formatting that is the bane of documents produced with WYSIWYG systems.

Sections 2 covers use of markup in the document. This includes formatting the text and, in Section 2.8, markup used to automate testing of code examples. Section 3 covers how to create the document. Section 4 covers the process to be followed in making changes to the document.

2 Formatting the Document

For compatibility with the widest variety of editors, text should be wrapped to fit with 80 columns. Edits should avoid reflowing text as this complicates identifying real changes in the document.

2.1 Basic Formatting

For the most part, text should not use TeX (or LaTeX) formatting commands. In particular, font or size changes should not be used. Formatting of MPI names and C and Fortran code is handled with the macros defined below. You may use `\emph` to *emphasize* text, as in `\emph{emphasize}`.

The command `\texttt` may be used for miscellaneous code for which there are no appropriate MPI macros.

LaTeX defines many environments and many others may be added to LaTeX. To preserve a uniform appearance, use only these environments or the ones specifically defined for the MPI standard in Section 2.2. Most of these are well-known; where there is some subtle feature, this is noted (e.g., between "center" and "centering").

array

center This environment should be used to center tables and text outside of figures.

centering This environment should be used to center figures (it does not add additional space around the figure).

description

displaymath

enumerate

eqnarray

example This environment should be used for example program fragments

figure Note that captions (with the `\caption` command) go below figures.

itemize

obeylines

tabbing

tabular

table Note that captions (with the `\caption` command) go above tables (tables themselves are created with the `tabular` environment).

verbatim

Verbatim An alternative form of verbatim that allows the use of TeX commands within the Verbatim environment.

To include a graphic image, use `\includegraphics`. This command can rescale the size of the image. A typical use (from the Collectives chapter) is

```
\includegraphics[width=4in]{figures/mycoll-fig2}
```

Figures should be providing in both Encapsulated Postscript (`eps`) and PDF (`pdf`) formats. Figures should be placed in the `figures` directory, not in the chapter's directory.

Explicit linebreaks should be avoided unless they are used where a linebreak is always required, such as at the end of a line of declarations. Use `\gb` (for "good break") to tell LaTeX where it may be better to break a line. The reason for using this is that if subsequent edits to the text change the flow of the paragraph, the line breaks will be adjusted accordingly.

References to section, table, figure, or enumerated list items should not use the number that appears in the document. Instead, they should make use of the TeX commands `\label` and `\ref`. The `\label` command creates a symbolic label for the most recent command that creates a number. For example,

```
\section{Communication Calls}
\label{sec:onesided-putget}
```

creates the label `sec:onesided-putget`. This section can then be refered to with the `\ref` command:

```
Section~\ref{sec:onesided-putget} describes the ...
```

The `\label` command may also be used after a `\caption` command to get the number of a figure or table, and after the `\item` command in an enumerated list to get the number of that item. This approach is used in the One-Sided Communications chapter to refer to the different RMA rules. Using the `\label` and `\ref` commands ensure that the references remain correct even if a new numbered item is inserted into the document.

2.2 MPI Environments

users Advice for user

rationale Rationale for a choice in the MPI standard.

implementors Advice for implementers

constlist A list of MPI constants.

funcdef The environment used for many MPI function definitions. There are related environments `funcdef2` and `funddefna`.

2.3 MPI Objects

These macros are used to mark MPI objects in the code. In some cases, they provide automatic indexing for the names; all of these are rendered in a consistent font.

`\mpifunc` Used to refer to MPI functions in normal text, when the abstract, language independent function is meant. Because the name is language independent, it should be in uppercase. Example use: `\mpifunc{MPI\_BCAST}`.

The name `\func` is a deprecated alias. Do not use and consider replacing with `\mpifunc`.

`\cfunc` Like `\mpifunc`, but for C functions.

`\ffunc` Like `\mpifunc`, but for Fortran functions.

`\mpiarg` Use this to refer to (language independent) function arguments in normal text. E.g.: `\mpiarg{array\_of\_handles}`.

`\carg` Like `\mpiarg` but for C functions

`\farg` Like `\mpiarg` but for Fortran functions.

`\type` Use this to refer to the type of (language independent) function arguments in normal text. E.g.: `\type{MPI\_INTEGER}`

`\ctype` Just like `\type`, but for C types. E.g.: `\ctype{double}`

`\ftype` Just like `\type`, but for Fortran types. E.g.: `\ftype{INTEGER}`

`\const` Use this to refer to mpi-defined constants, in a language-independent way. Takes one argument, the name of the constant.
E.g.: `\const{MPI\_MINLOC}`

`\constskip` Like `\const` but does not put the name in the index.

Obsolete commands which may be encountered but should not be used include

`\mpifunci` Like `\mpifunc`, but meant to be used with functions defined in MPI-1.

2.4 Standard Names

The following ensure that the name "MPI" is in the proper font and size. The / that follows the name is used to ensure that spaces are preserved (otherwise, TeX removes the blanks after a TeX command from the output).

\MPI/ or \mpi/ Gives you MPI with the correct font. It is used to refer to MPI in general. Example usage:
`It is highly desirable that \MPI/ not use...`

\MPIII/ or \mpiii/ Use like `\MPI` but when you want to specifically refer to MPI-2.

\MPIIDOTO/ or \mpiidoto/ Use like `\MPI` but when you want to specifically refer to MPI-1.0.

\MPIIDOTI/ or \mpiidoti/ Use like `\MPI` but when you want to specifically refer to MPI-1.1.

\MPIIDOTII/ or \mpiidotii/ Use like `\MPI` but when you want to specifically refer to MPI-1.2.

\MPIJOD/ or \mpijod/ Use like `\MPI` but when you want to specifically refer to MPI-JOD.

\MPIRT/ Use like `\MPI` but when you want to specifically refer to MPI/RT (real time).

2.5 Defining MPI Functions

The environment `funcdef` is used to define the language-independent form of an MPI function. `\begin{funcdef}` takes one additional argument, like this:

```
\begin{funcdef}{MPI\_BCAST( bd\_handle, root, comm )}
```

This is then followed by one more more `\funcarg` commands. `\funcarg` takes three arguments: intent of the argument, the name of the argument, and a brief description of the argument. For example

```
\funcarg{\IN}{root}{rank of broadcast root}
```

The MPI notion of IN, OUT, and INOUT arguments is slightly different that in some programming language descriptions. See the description in Section 2.3 (Procedure Specification) of the MPI Standard for more details, but roughly, they are

`\IN` Argument (or the object to which the argument refers) is not changed.

`\OUT` Argument (or the object to which the argument refers) is an output result only.

`\INOUT` Argument (or the object to which the argument refers) is both input and output.

A complete example, simplified from that of `MPI_BCAST`, is

```
\begin{funcdef}{MPI\_FOO( bd\_handle, root, comm )}
\funcarg{\INOUT}{bd\_handle}{Handle to buffer descriptor.
   On root, this is the send buffer descriptor, elsewhere,
   this is the receive buffer descriptor.}
\funcarg{\IN}{root}{rank of broadcast root}
\funcarg{\IN}{comm}{communicator handle}
\end{funcdef}
```

2.6 Bindings

Bindings specify how an MPI operation or object is expressed in a particular programming language. These are best updated by following an example with a similar format.

2.7 Indexing

The MPI standard has five separate indices:

- Examples Index
- Constant and predefined Handle Index
- Declarations
- Callback Functions
- Functions

There is no separate "concept" index, though the examples index contains references to some important MPI concepts.

For the indices for constants, declarations, callbacks, and functions, most (if not all) of the entries are created by using the correct macros around the names:

Constants

```
\const{name}, \consti{name},\type{name},
\shorttype{name},
\info{name}, \infoval{name}, \error{name},
```

Declarations `\typedefindex{name}`

Callbacks

```
\mpitypedefbind{name}, \mpitypedefbindvoid{name},
\mpitypedefemptybind{name},
```

Functions

```
\begin{funcdef}[name], \begin{funcdef2}[name],
\begin{funcdefna}[name], \func{name}, \mpifunc{name},
\mpiemptybindidx{routine}{returnvalue}{indexed name},
```

In places where the above macros cannot be used (such as within verbatim environments, for examples, or in certain tables), the index entries may be added explicitly with:

Constants `\cdeclindex{name}` `\cdeclmainindex{name}`

Declarations `\typedefindex{name}`

Callbacks `\typedefindex{name}`

Functions `\mpifuncindex{name}, \mpifuncmainindex{name}`

(The callbacks are separated from the declarations when the indices are created).

For the examples index, entries must be added explicitly with
`\exindex{name}`

The best way to check the index entries for each chapter is to build the chapter separately (execute make in the chapter's directory) and check the index that is created for that chapter to ensure that all relevant entries are included. Inspection of the LaTeX file can also help, particular where

verbatim environments are used. A final check of the full index, looking for duplicates, misspelled routines, and missing entries, can help identify other problems.

For entries that have subitems, e.g., you want the index to look like

```
foo
   bar     27
   foobar  721
```

use an exclamation point, as in

```
\exindex{foo!bar}
...
\exindex{foo!foobar}
```

In most cases, you should not use either a hyphen or a comma in an index entry — instead, use the "!" form.

Many items should also have a primary index entry, particularly when the item has many index entries. The macros in `mpi-macs.tex` contain macros `\mpifuncmainindex` and `\cdelcmainindex` which do this. You can make an item a "primary" by adding `|uu` to the entry, as in

```
\index{foo|uu}
```

Do not use text formatting commands in index entries. There is a way to do that, but it must be done with some special commands (otherwise the sorting of the index entries may be done by the formatting command rather than the text of the item, which has happened to at least one publisher).

2.8 Code Examples

To reduce the number of errors in code examples, we use a tool that extracts code from the document and attempts to compile the code. Below is a code example (slightly modified) from the Point-to-point chapter.

```
\begin{example}
\label{pt2pt-exA}
\exindex{MPI\_SEND}
\exindex{MPI\_RECV}
\exindex{Datatypes!matching}
Sender and receiver specify matching types.
%%HEADER
```

```
%%LANG: FORTRAN
%%FRAGMENT
%%DECL: integer comm, rank, ierr, tag, a(2), b(2)
%%DECL: integer status(MPI_STATUS_SIZE)
%%ENDHEADER

\begin{verbatim}

CALL MPI_COMM_RANK(comm, rank, ierr)
IF (rank.EQ.0) THEN
  CALL MPI_SEND(a, 10, MPI_REAL, 1, tag, comm, ierr)
ELSE IF (rank.EQ.1) THEN
  CALL MPI_RECV(b, 15, MPI_REAL, 0, tag, comm, status, ierr)
END IF

\end{verbatim}
```

The `\exindex` command adds its argument to the index of examples. TeX comments between `%%HEADER` and `%%ENDHEADER` are used to provide extra information needed to compile the code, such as specifying the language (e.g., `%%LANG: C` or `%%LANG: FORTRAN`), declarations for variables (e.g., `%%DECL: int a;`), and whether the code is a complete routine or a fragment (with `%%FRAGMENT`). Look at other examples in the text or the file `README` in directory `mpicompilechk`.

To check the code examples in the MPI document, run `make check`.

2.9 Mathematics

TeX was originally designed to typeset mathematics and no system is as effective at correctly handling all of the requirements of mathematical typesetting. You may use any of the usual LaTeX or TeX mathematics formatting commands when typesetting mathematics (most of the mathematical formulas are in the Datatype sections). If you have specific questions, either consult any of the LaTeX documentation or the document master.

3 Building the Standard

To build the standard, simply use `make`. There are additional targets that may be used to build special versions of the standard.

clean Clean the directory tree of auxiliary files created by running make.

veryclean Like clean, but cleans more created files, including converted graphics files.

distclean Like veryclean, but also removes the document PDF files.

check Runs the utility to check the code examples. This utility must have been configured using the `configure` command in the directory `mpicompilechk`.

eachchap Builds each chapter separately. This is good for editing individual chapters, particularly for index entries and bibliographic references.

cleandoc Produce a clean version of the document with no markup about the changes in the standard (no change in font color, no changebars, not old/new text).

cleanbwdoc Like `cleandoc`, but in black and white only.

reviewdoc

reviewchangeonlydoc

bookprintingdoc

allversions Use this to build all of the versions of the document that are made available

HTMLVERSION Create an HTML version of the standard. This uses the tool `tohtml`, which must be installed from `ftp.mcs.anl.gov/pub/sowing`. The better-known `latex2html` is (at last test) unable to handle a document of the size of the MPI standard.

When this target is used, check the file `latex.err` to see what problems were encountered in creating the HTML version.

BWHTMLVERSION Like `HTMLVERSION`, but in black and white only.

3.1 Building Individual Chapters

To build a single chapter, first build the full standard. This will provide the information necessary to include the proper values for references to sections in other chapters, and the correct chapter number. Then `cd` to the directory of the chapter and use `make`.

4 Editing Process

The MPI Document is developed by chapter subcommittees. Each chapter has a chair and a committee of at least 4 (including the chair). The committee is responsible for editing the document.

Changes in the document must be marked. The macros for this are presented below.

Changes may be introduced in two ways. The first is with the ticket system; this system is akin to a bug report system against the document. These can range from minor errors such as misspellings to the introduction of new routines. A ticket provides very specific details about the change, including the specific locations where changes are to be made. Changes that are made in response to a ticket must include the ticket number in the change macro.

The second way to introduce a change is for the chapter subcommittee to edit the chapter directly. This is appropriate for more extensive changes, which may range from thorough spelling and grammar corrections to introduction of significant new capability through new functions. The chapter committee may choose to have the MPI Forum vote on parts of this process separately, for example, a new function may be brought before the Forum for a vote. However, a final decision is based on a vote for the chapter as a whole (as well as a vote on the standard as a whole).

Note that all changes must be approved with the MPI Forum's process. At this writing, this requires a reading of the chapter, followed by two successful (majority) votes, each reading and vote held during a different meeting (this is intended to give adequate time for reflection and review).

This process differs from the MPI-2.1 and MPI-2.2 process because those versions were intended as minor edits to the standard. The process described above follows the process used for MPI-1 and MPI-2, where chapters were written by subcommittee, with frequent feedback from the MPI Forum in terms of straw votes or Forum votes on particular features, but without Forum votes for each individual change.

Minor edits Minor edits consist of corrections in spelling, grammar, and use of TeX/LaTeX macros. These must be marked as changes with either the relevant ticket number or with a pseudo ticket number of 0, which indicates a change made by the chapter committee.

Major edits Major edits consist of anything that is not a minor edit. These must be marked as changes; the ticket number should be used if available, otherwise use 0.

4.1 Update Macros

The macros in this section should be used to mark changes in the document. The most general form is this macro:

```
\MPIupdateBegin{3.0}{ticket-number}
... changed text
\MPIupdateEnd{3.0}
```

A short form,

```
\MPIupdate{3.0}{ticket-number}{update}
```

may be used for very short changes, and

```
\MPIreplace{3.0}{ticket-number}{old text}{new text}
```

for replacements of text.

Deletions should be marked (where possible) with

```
\MPIdeleteBegin{3.0}{ticket-number}
% ... deleted text, commented out in LaTeX
% % ... comment out comments as well
\MPIdeleteEnd{3.0}
```

Note that the text to be deleted is commented out using the TeX comment symbol. This is a pragmatic choice - it is possible to force TeX to ignore blocks of text, but if those blocks of text contain certain TeX or LaTeX commands, the deletion may not work properly. This approach, while less elegant, is more certain.

A short form,

```
\MPIdelete{3.0}{ticket-number}{text to delete}
```

may be used for short deletions.

If the `\MPIdelete...` form cannot be used, then removed or replaced text should be commented out if at all possible

While a chapter is being developed by a chapter committee, that committee may choose to mark updates, changes, or alternatives in other ways. This is acceptable as long as when the chapter is presented to the MPI Forum, the update macros described above are used.