

# Backward Compatibility WG

3/9/2010 update

# Current Efforts

- API modifications
  - How to handle changes in API's
- MPI\_Count
  - How to handle this specific issue
- Advice to ticket authors
  - General advice on API and functionality interaction with pre-existing API's and code

# Current Efforts

- Attempt #1 – “HDF style”
  - HDF versioning approach resisted by the forum
    - PMPI interface concern
- Attempt #2 - Simple naming convention
  - Define **changed**: A new API is introduced which is intended to supersede an existing API (the previous version is removed, deprecated or discouraged)
    - Includes argument or functionality changes
  - Define **basename**: the basename of an API is the name of the API with any trailing integers removed
    - `echo "API name" | sed -e 's|\[A-Za-z_\]*\[0-9]*|1|g`
  - If an API is **changed**, the new API will use the previous basename with an integer postfix which represents the version of the standard where the new version is first introduced.
  - For MPI-3 the post pended integer will be 3

# Draft text

As the MPI standard evolves, there are times where it is desirable to change the parameter list for a particular function in the standard. To preserve backward compatibility with existing applications, a new name is chosen for the new version of the API. This has occurred in the past when, for example, `MPI_COMM_CREATE_KEYVAL` was created to supersede `MPI_KEYVAL_CREATE`. In an effort to prepare for future changes of this nature, MPI will use an integer tag appended to existing function names whenever it is necessary to create a different but unique name for an existing function. The tag reflects the MPI Standard version in which the API was first introduced. For this version of the standard, the integer 3 is appended to any functions which require a new name. For example, this version of the standard will introduce `MPI_Get_count3`.

**Feedback:** All but 1 of non-abstainers believed we should use descriptive names instead of version related names.

# Naming Convention, cont.

- User: Do I need MPI\_Send or MPI\_Send3?
- Answer#1: Look up in the version of the standard you are using and see what the most recent name is for this API
- User: Aghhh
- Answer #2: All API's should have the latest document appended to them
- Implementors, testors, PMPI Users: Aghhh

# Naming Convention, cont.

- Compromise?
  - An implementation may include versions of all API's that match the most recent version of the standard, but are not required.
  - MPI\_Send3 is not standardized and its use is non-portable and non-compliant
  - You can still use it if your favorite MPI includes it.
  - Win win? Lose lose?

Feedback: unanimously apposed  
to including the compromise

# Draft Text

Feedback: based on feedback to use descriptive names, this text becomes unnecessary

For API's which have not changed, an implementation may chose to include a version of the API that includes the "3" designation, but doing so is not required by the standard and code written to use this API is not portable across MPI implementations. Providing these unofficial interfaces, however, may be convenient to developers who want to use the most recent version of all API's without having to determine the version of the standard in which an API was most recently changed.

# Miscellaneous

- Clarifying unspecified functionality does not change version number
- “Large” changes will use a new API to make the functionality difference clear.
- Versioning number only appended to changed API's, not new API's

# Draft Text

If a version of the standard clarifies the behavior of an API which was previously undefined, the API version will not be incremented. Any non-trivial modification in the specification of an API will be handled by introducing a completely new API.

New API's which are introduced by the MPI Standard will not have an appended version designation. Only API's which have been modified since their initial introduction are required to use an appended version designation.

# MPI\_Count

- Consensus (? Trusting Jeff S.'s report as I wasn't here) on MPI\_Count:
  - Change API's only for MPIIO functions
  - Related querying functions must be changed as well.
  - New functions to use MPI\_Count
  - Must be at least 32-bit (signed)
  - Integer assignable (like MPI\_Aint)

# Draft Text

Feedback:  
MPI\_Get\_Count ->  
MPI\_Get\_count

The MPI-2.2 standard uses an int in C and an INTEGER in Fortran for MPI function arguments which communicate a number of data type elements between the caller and the MPI library. Examples include MPI\_Send, MPI\_Get\_Count, MPI\_Pack, MPI\_Type\_contiguous and MPI\_File\_write. The use of a signed 32-bit integer value limits the range of counts to values less than or equal to 2,147,483,648. As computing systems have grown in size, there are occasionally situations which require sending larger counts than allowed by a signed 32-bit integer. To accommodate such use, all new API's introduced in the MPI-3 standard that communicate the concept of a "count" will accept the type MPI\_Count. The implementation of MPI\_Count is determined by the MPI implementation with the restriction that it must be minimally capable of storing values from 0 to 2,147,483,648. Like MPI\_Aint, MPI\_Count must be capable of being assigned unsigned integer values. The MPI implementation should document the maximum integer value which can be assigned to MPI\_Count for that implementation.

Feedback: change 2<sup>nd</sup> to last sentence to reference "integer base type"

# Draft Text

In order to change the API for all calls which could possibly make use of MPI\_Count, name differentiation is necessary to allow for compatibility with existing MPI application code. This is particularly true when the API takes a pointer to memory which is intended to hold a count such as MPI\_Get\_count. Introducing name variations of existing functions is deemed costly in terms of the total number of API's that must be maintained, tested and documented. Therefore, existing MPI functions are being updated only when a clear need is evident. The MPI-3 standard changes the API's of MPI IO related functions which use a count argument. In addition, this requires that routines used to access values with MPI\_Status also be changed. These API changes are being handled as described in the section "Handling of API changes". The following new API's are being introduced which use MPI\_Count:

# API's changed

- MPIO API's
- The following additional API's:
  - MPI\_Status\_set\_elements
  - MPI\_Get\_count
  - MPI\_Get\_elements

# Advice to ticket authors

- In progress
- What should this cover?
- Recommendations vs. watchdog group