Backward Compatibility Guidelines for New MPI Routines

1) **Suffixes Added to Symbols.** Each backward compatibility change will introduce a single character that reflects some aspect of the reason for the change. All characters will be appended to unmodified symbol names after a single underscore character, with the characters in alphabetical order. For example, a change introduced for compatibility with large "count" parameters could use the character 'L', and routines affected by only this change will have a suffix of '_L'. If another backward compatibility change occurred which introduced the 'A' character for suffixes, routines affected by only the second change would use a suffix of '_A' and routines affected by both changes would use a suffix of '_AL'.

2) **New Routines Should Use the "Best" Parameters.** In order to be as future-proof as possible, all new routines added to the standard should use parameter and return types that reflect the current best practices for MPI routine design. In most cases, this will be the largest type possible that is appropriate for the parameter. For example, parameters that count values should use 'MPI_Count' instead of 'int'.

3) **New Symbols Should Use Unsuffixed Names.** In order to present as unified of an interface to application developers as possible, all new symbols added to the standard should not append any backward compatibility suffixes for the symbol names. For example, a new routine that used MPI_Count for a parameter should be called "MPI_FOO", not "MPI_FOO_L".

4) **New Functionality Will Not Introduce Compatibility Routines.** The logical corollary of the previous two guidelines implies that when new functionality is added to the standard, only the routines with the best parameters and no backward compatibility suffixes should be added to the standard.

5) **Unsuffixed Symbols Will Be Marked as 'Discouraged'.** When a suffix is added to a symbol, this indicates that the unsuffixed version of the symbol should no longer be preferred by application developers. The unsuffixed version of the symbol will be marked as 'discouraged' in the standard, indicating that developers should use the suffixed version. Discouraging use of a symbol does not indicate that the symbol will be deprecated in a future version of the standard, although it might be. Discouraging use of a symbol indicates that application developers who wish to adhere to "best practices" principles should avoid using the symbol in newly written code, and should migrate old code away from discouraged symbols when possible. (If a symbol has multiple suffixed versions, the version of the symbol with the most suffix characters will be the preferred version and all other versions will be discouraged)

6) **Encouraging Best Practices.** A high-quality implementation of the standard will have a mechanism for indicating use of discouraged routines to application developers. For example, an implementation may have a way to remove all discouraged symbols from the library when it is built (causing failures at link-time), or have a way to create a compile warning or error when a discouraged symbol is encountered in source code compiled using *mpicc*.