Doc.no.: P0921r2

Date: 2018-05-06

Reply-to: Titus Winters, Ashley Hedberg

Audience: LEWG, EWG

Preamble

We propose the following text be adopted as the basis for a new standing document. If accepted, we encourage all committee members to discuss it and reference it regularly. We believe that clarifying the line between "what will work for the moment" and "what may not work in the face of changes to the standard" will improve the quality of C++ code and reduce the difficulty in updating to new language versions.

Standard Library Compatibility

For a sufficiently clever user, effectively any change we make to the standard library will be a breaking change. In a few instances, the standard itself is clear about what users are not allowed to do ("do not take the address of member functions of standard types", "do not add new names into namespace std") - those restrictions are not widely understood, are incomplete, and leave completely unclear why those rules are in place. In keeping with general discussions of compatibility, and as mentioned in P0684, we should be clear to users about what does and does not constitute supported use of the C++ standard library. What rights does the committee reserve in order to make changes to the standard library in the future?

Note in particular that users of the standard are not special in this: abuse of these rules is generally bad usage for most libraries.

Rights the Standard Library Reserves for Itself

Primarily, the standard reserves the right to:

- Add new names to namespace `std`
- Add new member functions to types in namespace `std`
- Add new overloads to existing functions

- Add new default arguments to functions and templates
- Change return-types of functions in compatible ways (void to anything, numeric types in a widening fashion, etc).
- Make changes to existing interfaces in a fashion that will be backward compatible, if those interfaces are solely used to instantiate types and invoke functions. Implementation details (the primary name of a type, the implementation details for a function callable) may not be depended upon.
    - For example, we may change implementation details for standard function templates so that those become callable function objects. If user code only invokes that callable, the behavior is unchanged.