Deprecate Certain Declarations in the Global Namespace

Document #: WG21 P0657R2
Date: 2018-10-07
Project: JTC1.22.32 Programming Language C++
Audience: LEWG ⇒ LWG
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

C library facilities are provided by C++ headers **<c***name***>**. As in all C++ headers, these entities' names are declared within namespace **std**. Unlike other C++ headers, the **<c***name***>** headers may optionally also declare these same names within the global namespace. This paper proposes to deprecate these headers' declarations at global namespace scope.

But what about your true name? It is not necessarily your given name. But it is the one to which you are most eager to respond when called.

— VERA NAZARIAN

If you can't see past my name, you can't see me.

— DASHANNE STOKES

But he that filches from me my good name / Robs me of that which not enriches him, / And makes me poor indeed.

— WILLIAM SHAKESPEARE

1 Background

According to [headers]/3–4, "The facilities of the C standard library are provided" by the contents of the following headers,[1] collectively known as **<c***name***>** headers:

<cassert>	**<climits>**	**<cstddef>**	**<cwchar>**
<cctype>	**<clocale>**	**<cstdint>**	**<cwctype>**
<cerrno>	**<cmath>**	**<cstdio>**	
<cfenv>	**<csetjmp>**	**<cstdlib>**	
<cfloat>	**<csignal>**	**<cstring>**	
<cinttypes>	**<cstdarg>**	**<ctime>**	**<cuchar>**

[1] These headers, specified in Table 19, are related to, but distinct from, the now-deprecated "C standard library headers" specified in Table 136 and collectively known as **<***name***.h>** headers.

Let us briefly explore what namespaces are involved in declaring the names of these facilities:

- According to [contents]/2, "All library entities ... are defined within the namespace **`std`** or namespaces nested within namespace **`std`**."
- In addition, footnote 167 immediately notes that "the C++ headers for C library facilities (15.5.1.2) may also define names within the global namespace."
- More precisely, [headers]/4 specifies that "Except as noted [elsewhere], the contents of each header **`c`***name* is the same as that of the corresponding header *name*`.h` as specified in the C standard library (Clause 2). In the C++ standard library, however, the declarations ... are within namespace scope (6.3.6) of the namespace **`std`**. It is unspecified whether these names ... are first declared within the global namespace scope and are then injected into namespace **`std`** by explicit *using-declarations* (9.8)."
- Finally, [depr.c.headers.other]/4 provides a clarifying Example: "The header **`<cstdlib>`** assuredly provides its declarations and definitions within the namespace **`std`**. It may also provide these names within the global namespace."

In sum, these **`<c`***name*`>`** headers (a) declare all their entities' names to be members of **`std`**, and (b) may, but need not, additionally declare some or all their entities' names to be members of the global namespace. However, even if the **`<c`***name*`>`** headers declare none of their names in the global namespace, those names are nonetheless reserved therein:

- [extern.names]/3: "Each name from the C standard library declared with external linkage is reserved to the implementation for use as a name with **`extern "C"`** linkage, both in namespace **`std`** and in the global namespace."
- [extern.names]/4: "Each function signature from the C standard library declared with external linkage is reserved to the implementation for use as a function signature with both **`extern "C"`** and **`extern "C++"`** linkage, or as a name of namespace scope in the global namespace."
- [extern.names]/footnote 181: "The function signatures declared in **`<cuchar>`**, **`<cwchar>`**, and **`<cwctype>`** are always reserved, notwithstanding the restrictions imposed in subclause 4.5.1 of Amendment 1 to the C Standard for these headers."
- [extern.types]/1: "For each type T [itemized in footnote 182] from the C standard library, the types **`::T`** and **`std::T`** are reserved to the implementation and, when defined, **`::T`** shall be identical to **`std::T`**."

2 Discussion and proposal

Why are names from the **`<c`***name*`>`** headers reserved in the global namespace as specified above? Consider the following coding dilemmas for portable programs that **`#include`** one or more of these headers:

1. Such programs can't rely on having names from these headers available in the global namespace, as those names "may, but need not" be declared there.
2. At the same time, such programs must account for the possibility that names from these headers may have been declared in the global namespace and thus can possibly overload or collide with user-declared names in that namespace.

In other words, portable programs must pretend that these declarations are in the global namespace scope, but must avoid using them from there. The only way to use these names safely as user-declared names is to avoid any use of the standard library.[2]

[2] Any header may **`#include`** one or more of the **`<c`***name*`>`** headers,: thus, indirect inclusion is always possible.

Accordingly, to be safe, portable programs would have to treat all names declared in any **<c***name***>** header as reserved in the global namespace, even had [extern.names] and [extern.types] not already so specified. These seem unfortunate restrictions.

Footnote 173 already discourages programs from relying on these names in the global namespace:

> The "**.h**" headers dump all their names into the global namespace, whereas the newer ["**c***name*"] forms keep their names in namespace **std**. Therefore, the newer forms are the preferred forms for all uses except for C++ programs which are intended to be strictly compatible with C.

Taking the next step along this path, we now **propose to deprecate, in these <c*name*> headers, every entity's declaration within the global namespace scope**. Note that other C++ entities declared at global scope (notably **::operator new** and **::operator delete**) are not affected by this proposal. Names of macros are also unaffected, of course.

3 Proposed wording[3]

3.1 Adjust footnote 167 (attached to [contents]/2) as shown:

167) The C standard library headers (D.6) also define names within the global namespace~~,~~. ~~while~~ Although a deprecated practice, the C++ headers for C library facilities (15.5.1.2) may also define names within the global namespace.

3.2 Adjust [headers]/4 as shown:

4 Except as noted in Clause 20 through Clause 30 and Annex D, the contents of each header **c***name* is the same as that of the corresponding header **name.h** as specified in the C standard library (Clause 2). In the C++ standard library, however, the declarations (except for names which are defined as macros in C) are within namespace scope (6.3.6) of the namespace **std**. It is unspecified whether these names (including any overloads added in Clause 16 through Clause 30 and Annex D) are first declared within the global namespace scope and are then injected into namespace **std** by explicit *using-declarations* (9.8)[; however, declaring such names within the global namespace scope is a deprecated practice for any C++ header].

3.3 Adjust [headers]/9 as shown:

9 Annex K of the C standard describes a large number of functions, with associated types and macros, which "promote safer, more secure programming" than many of the traditional C library functions. The names of the functions have a suffix of **_s**; most of them provide the same service as the C library function with the unsuffixed name, but generally take an additional argument whose value is the size of the result array. If any C++ header is included, it is implementation-defined whether any of these names is declared in the global namespace. (None of them is declared in namespace **std**.) [However, declaring such names within the global namespace scope is a deprecated practice for any C++ header.]

[3] All proposed additions and ~~deletions~~ are relative to Working Draft [N4762]. Editorial notes are displayed against a gray background.

3.4 Adjust [res.on.headers]/1 as shown:

1 A C++ header may include other C++ headers, but should not include any C header. A C++ header shall provide the declarations and definitions that appear in its synopsis. A C++ header shown in its synopsis as including other C++ headers shall provide the declarations and definitions that appear in the synopses of those other headers.

3.5 Adjust [depr.c.headers.other]/2 as shown:

4 [*Example*: The header **<cstdlib>** assuredly provides its declarations and definitions within the namespace **std**. ItAlthough a deprecated practice, **<cstdlib>** may also provide these names within the global namespace. The header **<stdlib.h>** assuredly provides the same declarations and definitions within the global namespace, much as in the C Standard. It may also provide these names within the namespace **std**. —*end example*]

4 Addenda

4.1 Possible additional deprecations

We considered additionally deprecating the reciprocal relationship that allows **<***name***.h>** headers to "also provide these names within the namespace **std**." Since these headers are already deprecated per [depr.c.headers], it seems pointless to do so. However, we are willing to provide such wording if LEWG so directs.

4.2 Note for the future

If WG21 adopts this paper, we hope that at some future date the practice of injecting standard library names into the global namespace will become a thing of the past. To accomplish this goal will require that the already-deprecated **<***name***.h>** headers be excised from C++,[4] and that we then follow up on the measures proposed herein.

5 Acknowledgments

Many thanks to the readers of early drafts of this paper for their thoughtful comments.

6 Bibliography

[N4687] Richard Smith: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N4687 (post-Toronto mailing), 2017–07–30. http://wg21.link/n4687.

[N4762] Richard Smith: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N4762 (corrected post-Rappersville mailing), 2018–07–07. http://wg21.link/n4762.

[P0619R1] Alisdair Meredith, Stephan T. Lavavej, and Tomasz Kamiński: "Reviewing Deprecated Facilities of C++17 for C++20." ISO/IEC JTC1/SC22/WG21 document P0619R1 (pre-Toronto mailing), 2017–03–19. http://wg21.link/p0619r1.

[4] See [P0619R1] for another view.

7 Document history

Rev.	Date	Changes
0	2017-06-11	• Published as P0657R0, pre-Toronto.
1	2017-10-10	• Updated relative to post-Toronto Working Draft. • Cited paper by Meredith, et al. • Adapted citations to use wg21.link. • Incorporated minor editorial improvements. • Published as P0657R1, pre-Albuquerque.
2	2018-10-07	• Rebased on Working Draft [N4762]. • Published as P0657R2, pre-San Diego.