Doc. no.: X3J16/96-0089R1
WG21/N0907R1
Date: 6 July 1996
Project: Programming Language C++
Reply to: Beman Dawes
beman@dawes.win.net

Clause 17 (Library Introduction) Issues List - Version 7

History:

Initial: Distributed at the start of the Tokyo meeting.
Version 2: Distributed during the Tokyo meeting.
Version 3: Distributed in the post-Tokyo mailing. Reflects votes taken in Tokyo and issues added by the LWG sub-group.
Version 4: Distributed in the pre-Santa Cruz mailing.
Version 5: Distributed at the Santa Cruz meeting. Reflects resolution changes by the working group.
Version 6: Distributed in the pre-Stockholm mailing. Reflects votes taken in Santa Cruz and new issues added.
Version 7: Distributed at the Stockholm meeting. Reflects new issues and resolution changes by the working group.

<-->

Work Group: Library Clause 17
Issue Number: 17-001
Title: Header Inclusion Policy
Section: 17.3.4.1 [lib.res.on.headers]
Status: Open
Description:

The (original) header inclusion policy of allowing any C++ header to include any other C++ header creates portability problems for users. For example, the following might compile correctly with some implementations and fail with others:

```
#include <string>  // programmer meant to write <iostream>
using namespace std;
int main() {
   cout << "Hello, C++ World" << endl;
   return 0;
   }
```

The (current) header inclusion policy of specifying exactly which C++ headers include which other C++ headers causes difficulty for implementors. The worst case is when one header requires reference to another but the other is not specified as included. Another troublesome case occurs when the implementation of one header (like <complex>) could benefit from access to something in another header (like template numeric_limits in <limits>).

The LWG has discussed these problems at length, and explored several alternatives including implementor’s namespaces (rejected because they don’t allow template specializations by users). The approach discussed in Santa Cruz is to:

-- Put more effort into correcting the synopsis lists of required header #includes. Sensible inclusion minimizes both implementor’s and user’s problems.

-- Continue to require inclusion of headers listed in synopsis, but also allow inclusion of names from other headers. This allows a user to write portable programs (by only relying on names from the required headers), allows automatic diagnosis of non-portable programs, and yet also gives implementor’s access to names they want and need.

Resolution:

Change the Working Paper as follows:

- Add a new first paragraph to 17.3.4.1 [lib.res.on.headers]:

  A C++ header may include other C++ headers.

- Delete the bullet item in 17.3.4.2 [lib.res.on.macro.definitions] which begins:

  The C++ headers listed in Table 21, C++ Library Headers, shall include the header(s) listed in their respective Synopsis subclause…

- Delete the #include's from the respective Synopsis subclauses.

- Delete the editorial box from 17.3.4.1 [lib.res.on.headers].

Requestor: Beman Dawes
Owner:
Emails:
Papers:

<--->

Work Group: Library Clause 17
Issue Number: 17-016
Title: Remove C names from namespace std
Section: 17.3.1.2 [lib.headers]
Status: Open

Description:

Placing the C library in namespace std causes difficulty for some library implementors.

Resolution:

Change the Working Paper as follows:

In 17.3.1.2 [lib.headers], table 22, C++ Headers for C Library Functions, delete leading "c" from header names and append ".h" to header names.

In 17.3.1.2 [lib.headers], replace paragraph 4 with:

Except as noted in Clauses _lib.language.support_ through _lib.input.output_, the contents of each C++ header for the C library facility shall be the same as that of the corresponding header name.h, as specified in ISO C (Clause 7), or Amendment 1, (Clause 7), as appropriate.

In 17.3.1.2 [lib.headers], delete paragraph 5.

In Annex D, delete section D.4 [depr.c.headers]

Requestor: Cathy Kimmel Joly
Owner:
Emails: Library reflector messages 4598, 4599, 4600, 4601, 4602, 4603, 4604, 4605, 4606, 4607, 4608, 4609, 4610, 4611, 4614, 4615, 4618, 4619, 4620, 4621, 4622, 4623, 4624, 4625, 4626, 4628, 4630, 4632, 4633, 4634, 4635, 4636, 4638, 4639, 4640, 4641, 4643, 4645, 4646, 4647, 4650, 4651, 4652, 4653, 4654, 4655, 4656, 4662, 4663, 4664, 4666, 4676, 4689, 4690
Papers:

<--->

Work Group: Library Clause 17
Issue Number: 17-017
Title: Clarification of library derivation
Section: 17.3.4.7 [lib.derivation]
Status: Open

Description:

> The current WP only allows a C++ Standard Library class to be derived from another class only if it is a base class. This overly constrains implementors.
>
> The “as if” rule does not allow such derivation because it can be detected (see lib-4536).

Resolution:

Change the Working Paper section 17.3.4.7 [lib.derivation] as follows:

Add a new first paragraph:

> It is unspecified whether a class in the C++ Standard Library is itself derived from other classes (with names reserved to the implementation).

Delete the first bullet item which reads:

> It is unspecified whether a class in the C++ Standard Library as a base class is itself derived from other base classes (with names reserved to the implementation).

Requestor: John Max Skaller
Owner:
Emails: Library reflector messages 4529, 4530, 4532, 4534, 4536, 4538
Papers:

<--->

Work Group: Library Clause 17
Issue Number: 17-018
Title: Clarification of C function-like macros
Section: 17.3.1.2 [lib.headers]
Status: Open

Description:

The current WP’s description of C headers is unclear as to the treatment of macros. The current wording of 17.3.1.2 paragraph 4 is:

> Except as noted in Clauses 18 through 27, the contents of each header cname shall be the same as that of the corresponding header name.h, as specified in ISO C (Clause 7), or Amendment 1, (Clause 7), as appropriate. In this C++ Standard library, however, the declarations and definitions are within namespace scope _basic.scope.namespace_ of the namespace std.

Resolution:

Replace 17.3.1.2 paragraph 4 of the Working Paper with:

> Except as noted in Clauses 18 through 27, the contents of each header cname shall be the same as that of the corresponding header name.h, as specified in ISO/IEC 9899:1990 Programming Languages C (Clause 7) , or ISO/IEC:1990 Programming Languages – C AMENDMENT 1: C Integrity , (Clause 4), as appropriate, as if by inclusion. In the C++ Standard library, however, the declarations and definitions (except for names

which are defined as macros in C) are within namespace scope (_basic.scope.namespace_) of the namespace std.

Names which are defined as macros in C shall be defined as macros in the C++ Standard library, even if license is granted in C for implementation as functions. [Note: the names defined as macros in C include the following: assert, offsetof, va_start, va_arg and errno, setjmp and va_end.]

Names which are defined as functions in C shall be defined as functions in the C++ Standard library. [Note: This disallows the practice, allowed in C, of providing a "masking macro" in addition to the function prototype. The only way to achieve equivalent "inline" behavior in C++ is to provide a definition as an extern inline function.]

In 17.3.4.2 [lib.res.on.macro.definitions] remove the footnote regarding C "masking macros."

Requestor: Thomas Plum
Owner: Thomas Plum
Emails: lib-4688
Papers:

<-->

Work Group: Library Clause 17
Issue Number: 17-019
Title: C++ headers with .h forms
Section: (Annex D) D.4 [depr.c.headers]
Status: Open

Description:

[from lib-4548]

I have been looking at issue 17-007 of the Clause 17 Issues list which was accepted into the WP at the Santa Cruz meeting. This issue added fstream.h, iomanip.h, iostream.h and new.h to the list of C++ .h headers provided by the library so that existing programs will continue to work and do approximately the same things. The intent is for each of the above mentioned headers to include the corresponding C++ Standard version of the header followed by the using declaration for each symbol in the header.

This will not really provide a compatible solution. Unlike the .h versions of the C headers provided for compatibility the above mentioned headers have not been previously defined in a standard way. This means that the content of each of these headers will vary at least subtly in each existing implementation. Since the contents of the .h headers are non-Standard it is also difficult to determine how different their contents are from the existing C++ versions of the headers. For example <fstream>, <iomanip> and <iostream> are now templatized. In <new> the default declaration of new() throws an exception whereas the declaration in most existing versions of new.h would not.

It seems likely that providing C++ Standard versions of fstream.h, iomanip.h, iostream.h and new.h opens the door to lots of compatibility problems. The includer of the .h headers may reasonably expect their "old" implementation defined behavior not the C++ Standard one. It will also make it more difficult for library vendors to provide a completely backward compatible header file solution.

Resolution:

D.4 paragraph 4 currently reads:

The C++ headers

- <fstream.h>
- <iomanip.h>
- <iostream.h>
- <new.h>

are similarly available.

Option 1:

Change “are similarly available” to “are also supplied. The contents are implementation defined.”

Option 2:

Delete D.4 paragraph 4 entirely.

Requestor: Sandra Whitman
Owner:
Emails: lib-4548, 4556, 4557, 4561
Papers:

<-->

Work Group: Library Clause 17
Issue Number: 17-020
Title: Clarification of exceptions thrown by library
Section: 17.3.4.8 [lib.res.on.exception.handling]
Status: Open

Description:

In private email, Jonathan Schilling wrote:

```
>I looked at the complex class, which is completely silent
>on error handling (other than for the I/O operators).  Does this mean
>that
>
>        there are no error conditions, or
>
>        error conditions result in undefined behavior, or
>
>        errors are reported using the C math library method, or
>
>        errors will result in (predefined?) exceptions being thrown
```

Beman Dawes comments:

I think we want to give considerable flexibility to implementors. We would, however, like to encourage the reporting of errors by throwing exceptions, and encourage that those exceptions are from (or derived from) the standard exception classes.

Resolution:

Change the Working Paper 17.3.4.8 [lib.res.on.exception.handling] from:

Any of the functions defined in the C++ Standard library that do not have an exception-specification may throw any exception. An implementation may strengthen this implicit exception-specification by adding an explicit one.

To:

Any of the functions defined in the C++ Standard library that do not have an exception-specification may throw implementation-defined exceptions. An implementation may strengthen this implicit exception-specification by adding an explicit one.

Add a footnote:

Library implementations are encouraged (but not required) to report errors by throwing exceptions from (or derived from) the standard exception classes ([lib.bad.alloc], [lib.support.exception], [lib.std.exceptions]).

Requestor: Jonathan Schilling
Owner:
Emails:
Papers:

<-->

Dispositions:

17-002 Extending namespace std.
Closed in Tokyo by accepting the proposed resolution.
17-003 Violation of Requires preconditions.
Closed in Tokyo by accepting the proposed resolution.
14-004 Should namespace std be subdivided?
Closed in Santa Cruz without taking any action.
17-005 What does “extending namespace std” mean?
Closed in Santa Cruz by accepting the proposed resolution.
17-006 Action when program extends namespace std.
Closed in Santa Cruz by accepting the proposed resolution.
17-007 Which C++ headers have .h forms?
Closed in Santa Cruz by accepting the proposed resolution.
17-008 Relational operator templates.
Closed in Santa Cruz by accepting the proposed resolution.
17-009 Separate Library from Core Language in Document.
Closed in Santa Cruz without taking any action.
17-010 Too Many Classes and Features in Standard Library.
Closed in Santa Cruz without taking any action.
17-011 Library Defined in Terms of Templates.
Closed in Santa Cruz without taking any action.
17-012 Decouple Libraries.
Closed in Santa Cruz without taking any action.
17-013 How will users access non-ISO C symbols using C++ headers?
Closed in Santa Cruz by accepting the proposed resolution’s sections 1 and 4 only.
17-014 Requirements on compare functions.
Closed in Santa Cruz by accepting the proposed resolution.
17-015 Restrictions on macro definitions clarification.
Closed in Santa Cruz by accepting the proposed resolution.