Clause 18 (Language Support Library) Issues List - Version 6

Introduction

This document is a summary of the issues identified in Clause 18. For each issue the status, a short description, and pointers to relevant reflector messages and papers are given.

Active Issues

Work Group: Library Clause 18
Issue Number: 18-030
Title: Should operator new and delete be defined within the namespace std?
Sections: 18.4 Dynamic memory management [lib.support.dynamic]
Status: active
Description: Bill Gibbons in c++std-lib-4823

>17.3.1.1/2 says:
>
> All library entities shall be defined within the namespace std.
>
> Shouldn't this say "except operator new and operator delete"?
>
> And since this includes "size_t", the declarations of "operator new"
> in section 18.4 need to qualify "size_t", i.e.
>
> size_t => std::size_t

Proposed Resolution: Exclude operator new and operator delete from namespace std and change 17.3.1.1/2 to say: All library entities except operator new and operator delete shall be defined within the namespace std.

If this is the case then size_t needs to be qualified as std::size_t in 18.4, 18.4.1.1-18.4.1.3.

Requestor: Bill Gibbons
Owner: Sandra Whitman
Emails: c++std-lib-4823
Papers: None
A few days ago I got the question of whether signal handling and exceptions work together. The usual answer is 'no', but the question triggered a little more.

In C there is a big restriction of what you can do inside a signal handler. You cannot call any library function (with 3 exceptions) and you may not access or modify any global variable (except with type `volatile sig_atomic_t`).

These restrictions were needed to allow optimizers to ignore the possibility of signals.

In C++ we have inherited the signal function. So we have to check what restrictions are needed in C++. Regarding the common subset of C and C++ we can adopt the rules of C.

I first tried to get a list of things which are possible/not possible inside a signal handler. After some internal discussion I realized that even some very basic C++ constructs are critical. Two examples:

- Constructing a class object may put the address of the vtbl into the object. The equivalent code would not be strictly conforming in C.
- Declaring a variable with a destructor. In usual code this needs some adjustment so that the destructor will be called when an exception is encountered. In a portable implementation this would be done by pushing a description object on a global stack.

A second thought was whether we need to restrict only executed code or also potential executed code. As an optimizer may bundle all descriptions for EH into a single object even that may be critical.

So I would like to have a rule along the lines of:

A function registered as a signal handler may only do what it is entitled to do in the C standard. A function which uses (even potentially) a language or library feature not in C will cause undefined behaviour.

[Note: This also covers very minor additions!]

[Example:

```cpp
    inline void f(){} // inline is no C
    void g(int) { if (0) f(); } // g uses a non-C feature
    signal( SIGINT, &g ); // undefined behaviour
}
```

Although f is never called, activating a SIGINT causes undefined behaviour.

Note that using exception handling or RTTI would most probably cause problems on some machines.]

I know this rule is overly restrictive. On the other hand trying to figure out what really is possible inside a signal handler will need too much...
>time. In C the rule is: The only thing you can portably do is setting
>a global flag. My rule will keep that rule and allow an
>implementation
>to mostly ignore the possibility of signals.
>
> I think -core is the right group to discuss this because we mostly
>have to
>judge language features. (Besides, I don’t read -lib :-)
>
The result of this discussion should go into another paragraph in
section
>[lib.support.runtime] 18.7. Even if this topic is seemed to be too late

>for the Hawaii meeting I would like to get a technical response. In
>my view this is important enough to come up as a NB comment. I would
>rather like to raise a NB comment which was already agreed on
>technically.

Proposed Resolution:

Add a rule to section 18.7 [lib.support.runtime] describing
the behavior of signal handlers in C++. The rule would be
 something like:

A function registered as a signal handler may only do what it is
entitled
to do in the C standard. A function which uses (even potentially) a
language or library feature not in C will cause undefined behaviour.

[Note:
[Example:

    inline void f(){}   // inline is not C
    void g(int) { if (0) f(); }   // g uses a non-C feature

    signal( SIGINT, &g );   // undefined behaviour

] Although f is never called, activating a SIGINT causes undefined
behaviour.
]

Requestor: Erwin Unruh, erwin.unruh@mch.sni.de
Owner: Sandra Whitman
Emails: c++std-lib-4963, c++std-core-7122-c++std-core-7124
Papers: None

-------------------------------------------------------------------------
Work Group: Library Clause 18
Issue Number: 18-032
Title: Macros as reserved words
Sections: 18.1 [lib.support.types], 18.7 [lib.support.runtime]
Status: active
Description: Nathan Myers in c++std-lib-4892

In general this is a Clause 17 issue. Since some of the macros
in question are described in Clause 18 I added it here as well.

In response to reflector mail c++std-lib-4799-c++std-lib-4804
discussing whether errno is a reserved word or not, Nathan wrote:

> About errno: most readers don’t seem to realize that it is
> not only permitted, but required, for errno to be a macro (17.3.1.2).
I recognize that this doesn't apply to Fergus's question, because the macro is (formally, if not practically) defined only if <errno.h> or <cerrno> is #included.

Therefore, any object named "errno", or likewise "assert", "setjmp", "offsetof", "va_start", "va_end", or "va_arg", would be a big mistake, because real programs #include all kinds of things.

We should probably claim all of these as reserved words in all contexts, and be done with it.

Proposed Resolution:

Add text to 18.1 [lib.support.types] and 18.7 [lib.support.runtime] or Clause 17 if that is more appropriate indicating that "assert", "setjmp", "offsetof", "va_start", "va_end" and "va_arg" are reserved words.

Requestor: Nathan Myers, ncm@mill.cantrip.org
Owner: Sandra Whitman
Emails: c++std-lib-4892, c++std-lib-4799-c++std-lib-4804
Papers: None

---

Work Group: Library Clause 18
Issue Number: 18-033
Title: direct calls to terminate() and unexpected()
Sections: 18.6 [lib.support.exception]
Status: active
Description: Jonathan Schilling in c++std-lib-5116

The question of whether direct user calls to terminate() and unexpected() should be allowed was settled in the affirmative in Stockholm, by closing library issue 18-015 with no action. But because some WP wording implies that they are only called by the implementation, and because the semantics of direct-called unexpected() aren't defined, I think some WP changes are necessary.

An implementation-called unexpected() must either throw an exception, which the implementation will either let through or turn into bad_exception (depending on the violated exception specification), or terminate the program. What should the restrictions be on a direct-called unexpected()? Since the main purpose of direct calls is for simulated testing of possible error conditions, it seems to me that direct-called unexpected() should be allowed to throw any exception, or must terminate the program. An alternative would be to only allow it to throw bad_exception or terminate, but that gives less flexibility for testing. Of course if a direct-called unexpected() tries a rthrow, terminate() will get called, as no throw is active. (To simulate a rthrow, a manual throw of bad_exception can be made from unexpected()).

Accordingly, I propose the WP changes attached.
(see Proposed Resolution:)
Proposed Resolution:

18.6.2.2 Type unexpected_handler [lib.unexpected.handler]

<change first bullet in 'Required behavior' to>

--throw an exception that satisfies the exception specification
  (however, if the call to unexpected() is from the program rather than
  from the implementation, any exception may be thrown);

18.6.2.4 unexpected [lib.unexpected]

<replace existing section with>

void unexpected();

1 Called by the implementation when a function exits via an exception not
  allowed by its exception-specification (_except.unexpected_). May also
  be called directly by the program.

Effects:
  Calls the unexpected_handler function in effect immediately after
  evaluating the throw-expression (_lib.unexpected.handler_), if called
  by the implementation, or calls the current unexpected_handler
  function,
  if called by the program.

18.6.3.3 terminate [lib.terminate]

<replace existing section with>

void terminate();

1 Called by the implementation when exception handling must be abandoned
  for any of several reasons (_except.terminate_). May also be called
directly by the program.

Effects:
  Calls the terminate_handler function in effect immediately after
  evaluating the throw-expression (_lib.terminate.handler_), if called
  by the implementation, or calls the current terminate_handler function,
  if called by the program.

Requestor:      Jonathan Schilling, jls@sco.com
Owner:          Sandra Whitman
Emails:         c++std-lib-5116,c++std-lib-4918,c++std-core-7086,
c++std-core-7088
Papers:         None

Closed Issues
-----------------------------------------------------------------------------------

Issue Number: 18-001
Title:        Typedef typedef void fvoid_t(); not used anywhere
Last Doc.:    N0784=95-0184

Issue Number: 18-002
Title:        Redundant typedefs
Last Doc.:    N0784=95-0184

Issue Number: 18-003
Title: Call to set_new_handler() with null pointer
Last Doc.: N0784=95-0184

Issue Number: 18-004
Title: Inherited members explicitly mentioned
Last Doc.: N0784=95-0184

Issue Number: 18-005
Title: Call to set_terminate() or set_unexpected() with null pointer
Last Doc.: N0784=95-0184

Issue Number: 18-006
Title: <stdarg.h> and references
Last Doc.: N0784=95-0184

Issue Number: 18-007
Title: denormal_loss member to the numeric_limits class
Last Doc.: N0784=95-0184

Issue Number: 18-008
Title: global operator new
Last Doc.: N0784=95-0184

Issue Number: 18-009
Title: whither exception?
Last Doc.: N0784=95-0184

Issue Number: 18-010
Title: Exception specifications for class numeric_limits
Last Doc.: N0784=95-0184

Issue Number: 18-011
Title: Exception specifications for set_new_handler()
Last Doc.: N0784=95-0184

Issue Number: 18-012
Title: Exception specifications for set_unexpected() and set_terminate()
Last Doc.: N0784=95-0184

Issue Number: 18-013
Title: deleting a pointer obtained by a nothrow version of "operator new"
Last Doc.: N0784=95-0184

Issue Number: 18-014
Title: nothrow versions of "operator delete"
Last Doc.: N0784=95-0184

Issue Number: 18-015
Title: Should terminate() and unexpected() be in <exception> ?
Last Doc.: N0935R1=96-0117R1
Resolution: closed, no action (Stockholm)

Issue Number: 18-016
Title: numeric_limits and LIA-1/WG14/C Compliance
Last Doc.: N0935R1=96-0117R1
Resolution: closed, no action (Stockholm)

Issue Number: 18-017
Title: Run-time Dependent traps in numeric_limits
Last Doc.: N0935R1=96-0117R1
Resolution: closed, no action (Stockholm)
Issue Number: 18-018  
Title: Run-time Dependent Rounding in numeric_limits  
Last Doc.: N0935R1=96-0117R1  
Resolution: closed, no action (Stockholm)

Issue Number: 18-019  
Title: Extra Denorm Members in numeric_limits in Support of IEC 559  
Last Doc.: N0935R1=96-0117R1  
Resolution: closed, no action (Stockholm)

Issue Number: 18-020  
Title: numeric_limits static const int/bool Members Must be Constant Expressions.  
Last Doc.: N0935R1=96-0117R1  
Resolution: accepted proposal (Stockholm)

Issue Number: 18-021  
Title: Correction to nothrow in <new>  
Last Doc.: N0935R1=96-0117R1  
Resolution: accepted proposal 3 with modifications (Stockholm)

Issue Number: 18-022  
Title: Make nothrow a Type Instead of a Value.  
Last Doc.: N0935R1=96-0117R1  
Resolution: accepted as editorial change (Stockholm)

Issue Number: 18-023  
Title: Array Form of Operator delete[] Added to 18.4.1.2  
Last Doc.: N0935R1=96-0117R1  
Resolution: accepted as editorial change (Stockholm)

Issue Number: 18-024  
Title: Are Some numeric_limits static const Members Really Dynamic?  
Last Doc.: N0935R1=96-0117R1  
Resolution: closed, no action (Stockholm)

Issue Number: 18-025  
Title: Make references to throw references to throw() in 18.2.1  
Last Doc.: N0935R1=96-0117R1  
Resolution: accepted as editorial change (Stockholm)

Issue Number: 18-026  
Title: type_info from 95-0195/N0795  
Last Doc.: N0935R1=96-0117R1  
Resolution: rejected, no longer true (Stockholm)

Issue Number: 18-027  
Title: Describe rounding error  
Last Doc.: N0935R1=96-0117R1  
Resolution: accepted as editorial change (Stockholm)

Issue Number: 18-028  
Title: Type float_round_style edits  
Last Doc.: N0935R1=96-0117R1  
Resolution: accepted as editorial change (Stockholm)

Issue Number: 18-029  
Title: numeric_limits specializations example editorial changes  
Last Doc.: N0935R1=96-0117R1  
Resolution: accepted as editorial change (Stockholm)