Remove Deprecated Atomic Initialization API from C++26

Document #:	P3366R0
Date:	2024-06-23
Project:	Programming Language C++
Audience:	SG1, LEWG

Reply-to:	Alisdair Meredith <ameredith1@bloomberg.net>

Contents

1 Abstract

This paper proposes removing the deprecated atomic initialization facility from the next C++ Standard.

2 Revision History

R0 August 2024 (midterm mailing)

Initial draft of this paper, based on content in [P2863]

3 Introduction

The topic of this paper has been extracted from the general deprecation review paper, [P2863], into its own paper so as to better track its progress, since this topic has had a couple of reviews but is not reaching a real conclusion while embedded in the broader paper.

The original API to initialize atomic variables for C++11 was deprecated for C++20 when the `atomic` template was given a default constructor to correctly perform the necessary initialization — see [P0883R2] for details. This paper proposes that now is the right time to remove that API from the C++ Standard.

4 Analysis

This legacy API continues to function but is more cumbersome than necessary. No compelling case appears to be made that the API is a risk through misuse. However, if updating the C++ Standard's reference to the C Library up to C23 removes the `ATOMIC_VAR_INT` macro, we might want to consider its removal for C++26 as well.

While the `ATOMIC_VAR_INT` macro does no active harm, maintaining text in the standard always comes with a cost; for example, [P2866R4] required LWG time to review and update D.22.3 [depr.atomics.nonmembers].

The deprecation and removal of this feature is reflected in the C Standard that initially deprecated the `ATOMIC_VAR_INT` macro (marked it as obsolescent) in C17 and actively removed it from the C23 Standard, per [WG14:N2390]. WG21 should strongly consider removing this macro but perhaps as part of a broader paper to update our reference to the C23 Standard Library.

Note that the C standard retains a generic `atomic_init` function that is *not* part of C++; i.e., we do not support that generic function in `<stdatomic.h>`.

5 Design Principles

Remove deprecated features from the Standard specification at the earliest *practical* opportunity to minimize the burden of accumulating obsolete specifications to maintain, reference, distract, and teach (to avoid).

6 Proposed Solution

Remove the deprecated Standard Library API from C++26 while granting vendors permission to continue supplying it as a conforming extension, for as long as they desire, through the use of zombie names.

Note that, assuming [P2866R4] lands, which is ahead of this paper in the pipeline to plenary, then this paper will remove the remaining parts of D.22 [depr.atomics], so we will present wording assuming that paper will have landed. If that paper fails to proceed, then the only change to the wording would be that the parent clause D.22 [depr.atomics] is not removed.

7 C++26 Reviews

Pending.

8 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N4986], the latest draft at the time of writing, and for purposes of parallel merges, assumes that [P2866R4] or its latest update has been applied.

8.1 Add new identifiers to 16.4.5.3.2 [zombie.names]

16.4.5.3.2 [zombie.names] Zombie names

1 In namespace `std`, the names shown in Table 38 are reserved for previous standardization:

Table 1: Table 38 — Zombie names in namespace std [tab:zombie.names.std]

		pointer_to_binary_function
ATOMIC_VAR_INIT	declare_reachable	
	generate_header	pointer_to_unary_function
atomic_init		
auto_ptr	get_pointer_safety	ptr_fun
auto_ptr_ref	get_temporary_buffer	random_shuffle
binary_function	get_unexpected	raw_storage_iterator
binary_negate	gets	result_of
bind1st	is_literal_type	result_of_t
bind2nd	is_literal_type_v	return_temporary_buffer
binder1st	istrstream	set_unexpected
binder2nd	little_endian	strstream
codecvt_mode	mem_fun1_ref_t	strstreambuf
codecvt_utf16	mem_fun1_t	unary_function
codecvt_utf8	mem_fun_ref_t	unary_negate
codecvt_utf8_utf16	mem_fun_ref	uncaught_exception
const_mem_fun1_ref_t	mem_fun_t	undeclare_no_pointers
const_mem_fun1_t	mem_fun	undeclare_reachable
const_mem_fun_ref_t	not1	unexpected_handler
const_mem_fun_t	not2	wbuffer_convert
consume_header	ostrstream	wstring_convert
declare_no_pointers	pointer_safety	
~~declare_reachable~~		

8.2 Update Annex C

Wording for Annex C to come.

8.3 Strike from Annex D

D.22 [depr.atomics] Deprecated atomic operations

D.22.1 [depr.atomics.general] General

1 The header `<atomic>` (33.5.2 [atomics.syn]) has the following additions.

```
namespace std {
  template<class T>
    void atomic_init(volatile atomic<T>*, typename atomic<T>::value_type) noexcept;
  template<class T>
    void atomic_init(atomic<T>*, typename atomic<T>::value_type) noexcept;
```

```
  #define ATOMIC_VAR_INIT(value) see below
}
```

D.22.3 [depr.atomics.nonmembers] Non-member functions

```
template<class T>
  void atomic_init(volatile atomic<T>* object, typename atomic<T>::value_type desired) noexcept;
template<class T>
  void atomic_init(atomic<T>* object, typename atomic<T>::value_type desired) noexcept;
```

1 *Constraints:* For the `volatile` overload of this function, `atomic<T>::is_always_lock_free` is `true`.

2 *Effects:* Equivalent to: `atomic_store_explicit(object, desired, memory_order::relaxed);`

D.22.4 [depr.atomics.types.operations] Operations on atomic types

```
#define ATOMIC_VAR_INIT(value) see below
```

1 The macro expands to a token sequence suitable for constant initialization of an atomic variable of static storage duration of a type that is initialization-compatible with value.

[*Note 1:* This operation possibly needs to initialize locks. —*end note*]

Concurrent access to the variable being initialized, even via an atomic operation, constitutes a data race.

[*Example 1:*

```
atomic<int> v = ATOMIC_VAR_INIT(5);
```

—*end example*]

8.4 Update cross-reference for stable labels for C++23

Cross-references from ISO C++ 2023

All clause and subclause labels from ISO C++ 2023 (ISO/IEC 14882:2023, *Programming Languages — C++*) are present in this document, with the exceptions described below.

container.gen.reqmts *see*
container.requirements.general

depr.arith.conv.enum *removed*
depr.atomics *removed*
depr.atomics.general *removed*
depr.atomics.nonmembers *removed*
depr.atomics.operations *removed*
depr.atomics.volatile *removed*
depr.codecvt.syn *removed*
depr.conversions *removed*
depr.conversions.buffer *removed*
depr.conversions.general *removed*
depr.conversions.string *removed*
depr.default.allocator *removed*
depr.istrstream *removed*
depr.istrstream.cons *removed*
depr.istrstream.general *removed*
depr.istrstream.members *removed*
depr.locale.stdcvt *removed*

depr.locale.stdcvt.general *removed*
depr.locale.stdcvt.req *removed*
depr.mem.poly.allocator.mem *see*
mem.poly.allocator.mem
depr.ostrstream *removed*
depr.ostrstream.cons *removed*
depr.ostrstream.general *removed*
depr.ostrstream.members *removed*
depr.res.on.required *removed*
depr.string.capacity *removed*
depr.str.strstreams *removed*
depr.strstream *removed*
depr.strstream.cons *removed*
depr.strstream.dest *removed*
depr.strstream.general *removed*
depr.strstream.oper *removed*
depr.strstream.syn *removed*
depr.strstreambuf *removed*
depr.strstreambuf.cons *removed*
depr.strstreambuf.general *removed*
depr.strstreambuf.members *removed*
depr.strstreambuf.virtuals *removed*
depr.util.smartptr.shared.atomic *removed*

mismatch *see* alg.mismatch

9 Acknowledgements

Thanks to Michael Park for the pandoc-based framework used to transform this document's source from Markdown.

Thanks to Lori Hughes for reviewing this paper.

10 References

[N4986] Thomas Köppe. 2024-07-16. Working Draft, Programming Languages — C++.
https://wg21.link/n4986

[P0883R2] Nicolai Josuttis. 2019-11-08. Fixing Atomic Initialization.
https://wg21.link/p0883r2

[P2863] Alisdair Meredith. Review Annex D for C++26.
https://wg21.link/p2863

[P2866R4] Alisdair Meredith. 2024-07-15. Remove Deprecated Volatile Features From C++26.
https://wg21.link/p2866r4

[WG14:N2390] Jens Gustedt. 2019-06-07. Remove ATOMIC VAR INIT.
https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2390.pdf